André TONIC
Edward AREVIAN
Philippe MILLET

LES MYSTERES DU X07

L'ASSEMBLEUR DU CANON X07

Club C7

LES MYSTERES DU X-07

LE CANON X-07
ET
L' ASSEMBLEUR

PAR

André TONIC
Edward AREVIAN
Philippe MILLET

CLUB C7 - 1985
33 AVENUE PHILIPPE AUGUSTE
75011 PARIS

LES AUTEURS

André TONIC , 19 ans , étudiant en Sciences Economiques , participe activement depuis plusieurs années à l'élaboration de nombreux logiciels commerciaux sur micro-ordinateurs aussi bien portables qu'individuels . En outre , il est co-auteur de plusieurs recueuils de programmation . Il est également président du CLUB C7 .

Edward AREVIAN , 23 ans , étudiant en électronique et en informatique, est conseiller technique au sein du CLUB C7 . Il s'intéresse tout particulièrement à l'architecture interne des micro-ordinateurs .

Philippe MILLET , 23 ans , a brillamment terminé ses études commerciales : il s'occupe de la trésorerie à l'intérieur du CLUB C7 . De plus , il a acquis une certaine expérience dans l'exploitation des langages évolués .

REMERCIEMENTS

Nous désirons adresser **nos plus vifs remerciements à** :

_ **Mrs De la Rüe du Can et De Carville** , responsables MARKETING de la société CANON FRANCE , pour leurs informations relatives au X-07 .

_ **Melle HELAS** , étudiante , pour sa participation à la composition de cet ouvrage .

_ **Mr ROUSSEAU** , maître assistant en sciences , pour les nombreux renseignements concernant le langage machine du X-07 qu'il nous a aimablement donnés .

Les AUTEURS

AVANT - PROPOS

Cet ouvrage s'adresse à toutes les personnes désireuses de programmer en ASSEMBLEUR Z-80 . Plus spécifiquement , les utilisateurs du CANON X-07 y découvriront les moindres secrets relatifs à leur machine : adresses , zone système complète , explications de la ROM , le T6834 ...

Ponctué par de nombreux exemples et applications détaillées , ce livre constitue la porte ouverte à l'univers mystérieux et attirant de l'ASSEMBLEUR CANON . Il permettra à tous les possesseurs du X-07 d'utiliser l'optimum des possibilités de leur micro portable et de ses périphériques .

PREFACE

OCTOBRE 2070 , QUELQUE PART à MICROCITY , LA NUIT ...

J'errais depuis plusieurs heures dans **MICROCITY** ... J'avais débarqué dans cette ville remplie de consoles informatiques depuis deux jours et je n'avais toujours rien découvert . Je commençais à me demander si je n'avais pas parcouru 7000 kilomètres pour rien ...

Avant d'aller plus loin , je voudrais me présenter . Mon nom est **CANDORE** et je suis en quête d'un vieux maitre , détenteur d'un savoir très particulier ... En effet , en cette année **2070** , tous les langages informatiques sont devenus standards et se sont fondus en un seul : **le LASPOTOLO** (sigle pour désigner le "**LA**ngage Standard **PO**ur **TO**us **L**es **O**rdinateurs") . Etant possesseur d'une antique machine datant des années 1980 , un **CANON X-07** (conservé dans la naphtaline !) , j'espère retrouver les traces d'un langage appelé **"ASSEMBLEUR"** ... Pourquoi ? Une ancienne légende dit : "Le langage ASSEMBLEUR permettait jadis de tirer parti au **maximun** des possibilités de son ordinateur . Quiconque est initié à ce fabuleux langage devient le maitre incontesté de sa machine ... Mais seul le sage **ASSEMBLUS** connait parfaitement l'ASSEMBLEUR et peut vous permettre de dompter à jamais votre micro ..." . Le seul problème inhérent à cette légende était l'endroit où se terrait le sage en question . Heureusement , après **deux ans** de recherches , j'ai retrouvé sa trace ... ou du moins , je le crois !!

Je me retrouve donc à MICROCITY en train de vagabonder ... Les cabarets battent leur plein ainsi que les boites de nuit . Je regarde machinalement ma montre atomique : 2 h 25 . J'hésite à rentrer à l'hotel quand tout à coup , une **petite lumière** attire mon oeil ... Une baie vitrée est illuminée d'une lumière quasi irréelle . Je m'apprête à repartir quand je perçois un appel télépathique m'obligeant à revenir sur mes pas et à rentrer dans cette étrange maison ...

La porte grince ... Sûrement l'une des dernières portes à gonds de toute la galaxie !!! Une faible lueur m'invite à rentrer dans une pièce où un homme d'un certain âge médite . Subitement , il se met à parler :

"Je savais que tu parviendrais jusqu'à moi ... j'ai perçu tes ondes cérébrales à l'entrée de la ville , il y a quelques jours . Tu seras probablement le dernier élève que j'aurais ... Tu as fait une longue route pour me retrouver afin d'apprendre le vieux langage "ASSEMBLEUR" . Eh oui , je suis **ASSEMBLUS**, le vieux sage de la légende ..."

Il se redresse , me fixe de son regard profond et joint ses deux mains . J'éprouve une sensation de béatitude mêlée d'un sentiment de profonde sécurité . J'ai compris qu'il va accéder à ma requête ... Il va me transmettre son savoir !

ASSEMBLUS se dirige lentement vers le fond de la pièce et fait pivoter un panneau sculpté . Il me fait signe de le suivre et je découvre une salle au centre de laquelle se trouve un **CANON X-07** ! Le micro a l'air en parfait état de fonctionnement et se trouve relié à plusieurs périphériques tels que des imprimantes , des écrans , des coupleurs optiques ...

Le sage effleure la touche ON/OFF du X-07 et toute la pièce se met à tracer , à dessiner , à vibrer , à se colorer , à chanter ... Il comprend mon étonnement et déclare :

"Tu as l'air très étonné CANDORE mais cela est tout à fait normal ... Le CANON X-07 reste la seule machine dans la galaxie à pouvoir être commandée en ASSEMBLEUR Z-80 . **La guilde informatique** ayant imposé son langage et ses normes , l'ASSEMBLEUR a été proprement éliminé et plus personne ne peut visiter les entrailles de sa machine , comme jadis ... J'appartiens à un autre temps , à une autre époque . Je sais que ma mort est proche mais je ne veux pas commencer mon dernier voyage sans avoir légué la pratique du langage ASSEMBLEUR à quelqu'un de noble . Si tu le désires , je peux t'apprendre ce langage pour que tu puisses , toi CANDORE , dompter , maitriser et ne plus être l'esclave de ta machine . Le CANON sera alors pour toi la machine idéale et tu pourras réaliser tout ce qu'il fait en ce moment ..."

J'acquiesce lentement et ASSEMBLUS me conduit sous un casque à positrons rapides permettant à n'importe quel être humain d'emmagasiner des milliers d'informations très rapidement . Je perçois le démarrage du disque optique et je m'endors , bercé par le doux ronronnement des données pénétrant mon cerveau ...

OCTOBRE 1985 , QUELQUE PART EN FRANCE , 4 h 15 ...

Je sursaute et je m'aperçois que je suis en sueur ... Drole de rève !! Décidément , l'ASSEMBLEUR me travaille! Je glisse un oeil vers mon radio-réveil : 4 h 17 . Je me lève et m'assieds à mon bureau ... Un CANON X-07 , une dizaine de feuilles de listing , une imprimante et trois crayons occupent toute la table .

Ah , si mon rève se réalisait un jour ... Etre détenteur du langage ASSEMBLEUR , comprendre les moindres particularités du X-07 et avoir visité ses recoins les plus secrets ! Cela fait plus de 15 jours que je n'ai pas avancé ...

Mais un détail attire mon attention ... J'ai complètement oublié d'ouvrir le paquet cadeau offert par Odile pour mon anniversaire !!

D'une main fébrile , je le décachète délicatement et , ô miracle , je découvre l'ouvrage traitant du **"CANON X-07 et de l'ASSEMBLEUR"** .

Best Seller depuis des mois , il est épuisé dans toutes les librairies car tous les utilisateurs du X-07 se l'arrachent !! En effet , depuis la parution de cet ouvrage , des milliers de possesseurs de CANON ont appris à mieux connaitre leur micro en l'exploitant de façon optimale grâce à l'ASSEMBLEUR . J'ouvre doucement ce magnifique livre et , en page de garde , une dédicace m'est adressée : "A Candore , le palais de la programmation supérieure vous est désormais accessible . Les AUTEURS" .

Mes angoisses sont désormais terminées et même sans "casque à positrons" , je suis certain de maitriser bientôt parfaitement mon CANON X-07 !

A. TONIC

INTRODUCTION

"J'aimerais goûter aux joies de l'ASSEMBLEUR , mais comment faire ? ..."

Cette supplique , nous l'avons entendu un nombre incalculable de fois : dans la rue , au téléphone , dans notre courrier , chez les revendeurs ... Bref , de toute la France et même de plus loin , cette plainte nous est parvenue comme un appel d'agonie . La larme à l'oeil mais la plume au chaud , nous avons décidé d'écrire un ouvrage traitant de l'**ASSEMBLEUR Z-80** , langage informatique ô combien méprisé ou adoré , selon les humeurs du micro-ordinateur utilisé .

Mais un détail revenait très souvent : **"Mon CANON X-07 ne veut pas me dévoiler ses secrets !!"** . Une fois de plus , revigoré par ce nouveau défi, nous avons puisé dans nos dernières forces pour rédiger la suite de l'ouvrage : le **"SOFT"** et le **"HARD"** du CANON **X-07** a été disséqué , retourné , découpé au Laser afin de fournir aux plus audacieux programmeurs les clés du Nirvana .

Les pages qui vont suivre ont pour objectif d'une part , de vous familiariser avec l'ASSEMBLEUR d'un micro-processeur Z-80 et d'autre part , de vous accompagner au coeur du X-07 . En effet , le CANON ayant comme "cerveau principal" un **NSC 800** (version CMOS , à faible consommation d'energie , du célèbre ZILOG 80) , il est indispensable à l'explorateur téméraire de connaitre le langage primaire de la "bête" . Puis , peu à peu , nous vous entrainerons sur les plaines du SILICIUM où règne l'empire des seigneurs "Adresses" , "Sous-processeur T6834" , "Zone système" , "Hard" , "MEM" ...

Mais avant d'entreprendre un tel périple , il vous est nécessaire de posséder au moins quelques concepts d'un langage évolué . Sur le X-07 , le BASIC étant résident , il serait souhaitable que vous saisissiez déjà le sens et l'action des fonctions usuelles telles "PRINT" , "GOTO" , "GOSUB" , "FOR...NEXT" ...

Sinon , à notre grand regret , le monstre baptisé "INSOMNIA" aura rapidement anéanti les dernières parcelles de sommeil restant à votre acquis .

L'ouvrage que vous allez dévorer des yeux est architecturé de telle manière qu'à l'instant du dénouement , vous serez passé maitre en l'art de dompter le CANON X-07 . En effet , après avoir maitrisé l'ASSEMBLEUR , vous jonglerez gaiement avec les mystères enfin mis à jour de votre machine préférée . Pour vous aider dans cette voie très convoitée , de nombreux exemples et applications pratiques vous seront exposés . Ceux-ci vous permettront de "domestiquer" les derniers concepts rebelles à votre intelligence tenace .

Enfin , quelques annexes vous renseigneront efficacement sur les derniers points utiles pour continuer votre nouvelle quête de la terre promise nommée "CANONLAND" .

Nous espérons que cet ouvrage sur le langage machine du CANON X-07 comblera tous les fanatiques et adorateurs de cette machine si prisée . Les portes du paradis leur sont désormais ouvertes ...

A. TONIC

SOMMAIRE

2ème PARTIE : LES MYSTERES DU X-07

3ème PARTIE : APPLICATIONS

ANNEXES

1ère PARTIE :

L'ASSEMBLEUR Z-80

GENERALITES

Avant de nous lancer dans l'ASSEMBLEUR pur , ménageons-nous une petite mise en condition avec quelques généralités . Elles vous permettront de glisser en douceur vers le monstre sacré que constitue le langage machine ...

1.1 DEFINITION d'un ORDINATEUR .

"COMPUTER , ORDINATEUR , MICRO , MINI ..." . Tous ces mots magiques désignent en fait un "simple" amas de circuits électroniques . L'ordinateur est une machine **ne sachant pas réfléchir elle-même** : elle sait seulement répéter , un millier de fois si nécessaire , une opération et exécuter un programme lui permettant de résoudre un problème donné . Effectivement , ce n'est déjà pas si mal que ça !!

Tout ordinateur possède un "cerveau" (**le micro-processeur**) et une **mémoire** plus ou moins importante , se mesurant en **Kilo , Méga ou Giga Octets** . L'exécution d'un programme quelconque (jeu , CALC , Gestion de fichiers ...) entraine toujours les mêmes conséquences : le micro-processeur effectue les opérations qu'il trouve dans la mémoire et communique les résultats aux organes d'**Entrée/Sortie** (Ecran vidéo , Imprimante , Mémoire de stockage ...) . Un schéma simplifié peut vous faire saisir l'ensemble cohérent que constitue un ordinateur (Voir figure 1) .

Bien qu'il puisse se présenter de manière disparate , l'ordinateur forme un "tout" indissociable . Par exemple , le micro-processeur ne peut fonctionner sans mémoire ou encore , aucun résultat ne peut être visualisé sans écran .

Plus précisément , nous pouvons "ausculter" l'intérieur de la mémoire , coeur de cet univers fascinant . En effet , le type et les performances du système utilisé dépendent du **format des ordres placés en mémoire** . De plus , la façon dont est relié le micro-processeur à ses périphériques est primordiale .

1.2 FORMAT des ORDRES et des DONNEES .

On peut se poser la question de savoir comment des chiffres peuvent être stockés dans la mémoire d'un ordinateur . En fait , l'ordinateur étant électroniquement constitué , **la présence ou l'abscence d'un courant électrique** peut être facilement détectée .

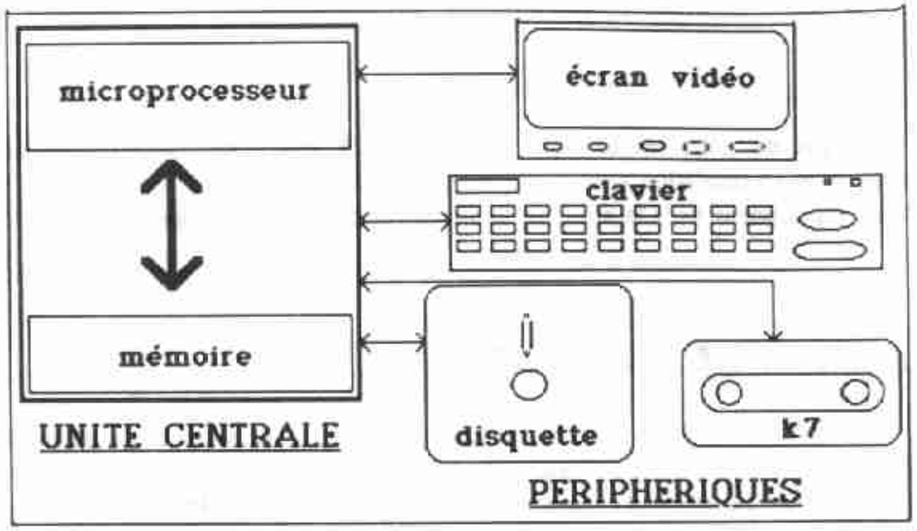


FIGURE 1 : SCHEMA d'un ORDINATEUR

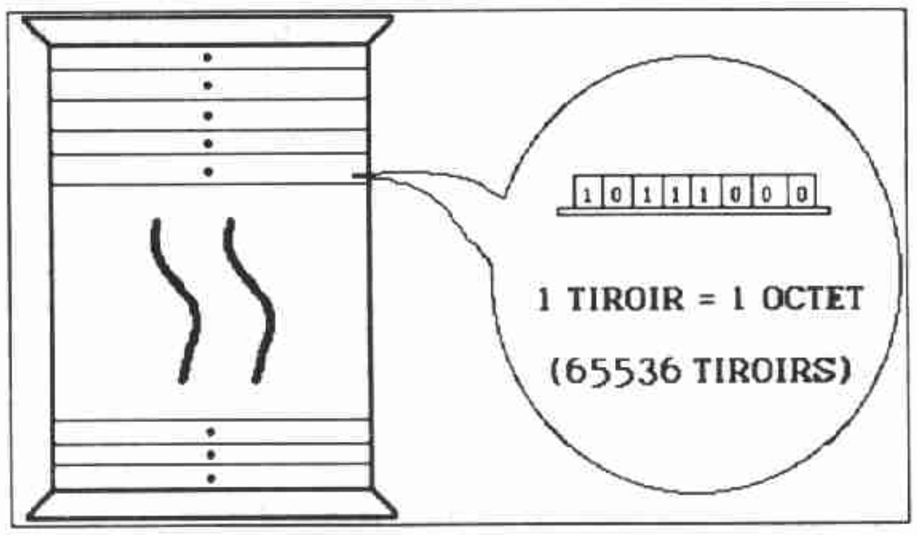


FIGURE 2 : SCHEMA de la MEMOIRE

Comme l'a déclaré en son temps DEMOCRITE : "l'ATOME est la plus petite partie de matière dissociable" . De même , une information de base nommée **BIT** , sera symbolisée par un "0" ou un "1" selon l'ouverture ou la fermeture du circuit . C'est également la plus petite partie de mémoire séparable .

Le micro-processeur du CANON X-07 , **le NSC 800** (National Semi-Conductor **800**) , est un **"8 bits"** . Il peut donc effectuer des opérations sur 8 bits simultanément . De plus , il est relié à la mémoire par un premier groupe de 8 fils appelé **"Bus de Données"** .

Mais quand vous demandez à une personne la capacité de mémoire de son X-07 , elle peut vous répondre : "Mon CANON contient 24 Kilo-octets de mémoire" . Que représente un **OCTET** ? Mettons fin à votre appréhension : un octet représente simplement **un groupe de 8 bits** , c'est à dire un groupe de 8 informations binaires . De même , quand on vous déclare : "Mon micro est un vrai 16 bits" , vous pouvez être sûr que cet ordinateur contient des octets de16 bits et non de 8 bits comme le X-07 .

Exposons maintenant nos souvenirs "binairiens" ...

1.3 Le BINAIRE .

Nous savons maintenant que l'ordinateur code les chiffres avec des "1" et des "0" : étrange ressemblance avec **la base 2** !!

Ainsi , sur 8 bits , nous pouvons coder en binaire les chiffres de 0 à 255 , soit 256 symboles . En utilisant le même principe qu'en base 10 , on donnera à chaque bit une valeur dépendante de sa position . Un exemple vous sortira du brouillard :

- 101 en base 10 est aussi représenté par : $1.1 + 0.10 + 1.100$
  Soit , puisque notre base est 10 : $1.1 + 0.1.10 + 1.10.10$
  Ou encore : $1.10^0 + 0.10^1 + 1.10^2$

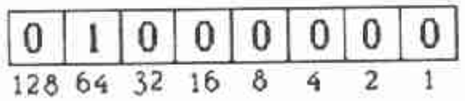
- 101 en base 2 est aussi représenté par : $1.1 + 0.2 + 1.4$
  Soit , puisque notre base est 2 : $1.1 + 0.2.1 + 1.2.2$
  Ou encore : $1.2^0 + 0.2^1 + 1.2^2$

(**N.B** : pour les décompositions en binaire , voir figure 3)

Sans le savoir , vous venez d'appréhender la notion de rang et de poids (Voir figure 4) . En effet , selon son emplacement (rang) , un même chiffre n'a pas la même valeur (poids) . De plus , puisque nos mots sont de 8 bits , ils auront la configuration définie à la page 19 .

DECIMAL	BINAIRE	HEXA.
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

FIGURE 3 : CORRESPONDANCES



POIDS : 0 ou 1

RANG : UNE DES CASES DESSINEES

FIGURE 4 : DEUX OCTETS DIFFERENTS

Un mot ou un octet : $a.2^7 + a.2^6 + a.2^5 + a.2^4 + a.2^3 + a.2^2 + a.2^1 + a.2^0$

Soit avec a=1 : $1.2^7 + 1.2^6 + 1.2^5 + 1.2^4 + 1.2^3 + 1.2^2 + 1.2^1 + 1.2^0$
$= 255$

Donc , le nombre 255 représente la valeur maximale qui peut être attribuée à un octet .

1.4 La MEMOIRE .

Puisque le micro-processeur travaille avec des mots de 8 bits , la mémoire ne contiendra que des mots de 8 bits également . Elle est reliée au NSC 800 (pour le CANON X-07 ...) par un deuxième bus de 16 fils . Ce **"bus d'Adresse"** sert à indiquer quels octets elle doit fournir au micro-processeur . En adoptant la notation binaire , 16 fils correspondent à $2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$ = **65536 cases désignables possibles** .

En fait , on peut assimiler la mémoire du X-07 à un meuble de 65536 tiroirs (les octets) , numérotés de 0 à 65535 , chaque tiroir contenant 8 petites cases (les bits) . Vous pouvez vous référer à la figure 2 pour mieux cerner cette métaphore .

1.5 L' HEXADECIMAL .

65536 positions possibles !! Cela parait bien difficile à désigner avec des 1 et des 0 ! Pour pallier à cet inconvénient , une notation intermédiaire fut adoptée entre le binaire et le décimal : l'hexadécimal .

Si l'on désire coder des nombres de 4 bits , on pourra coder 16 symboles comme sur la figure 3 .

Vous remarquerez qu'après le chiffre 9 en hexadécimal , on découvre la lettre "A" . Les mathématiciens , devant statuer sur des symboles quelconques, ont décidé que les lettres A à F désigneraient les nombres de 10 à 15 .

Maintenant , nous savons transformer un mot de 4 bits en hexadécimal ... Mais comment procède t-on pour un mot de 8 bits ? Surtout ne vous défenestrez pas pour si peu car , en fait , un mot de 8 bits est constitué par deux mots de 4 bits ! Afin de vous prouver la simplicité de la méthode , prenons deux exemples :

_ "10101111" en binaire peut être facilement transcrit en hexadécimal . En effet , la première tranche "1010" constitue 4 bits et la deuxième tranche "1111" constitue 4 autres bits . Par conséquent , en s'aidant de la figure 3 , "1010" équivaut à "E" et "1111" équivaut à "F" . En définitive , "10101111" équivaut à "EF" en hexadécimal !

_ De même , "10111000" équivaut à "B8" en hexadécimal car "1011" est équivalent à "B" et "1000" à "8" . Simple , non ?

De plus , signalons que la technique est la même pour un mot de 16 bits . Par conséquent , "1111111111111111" équivaut à "FFFF" en hexadécimal .

La notation hexadécimale est beaucoup plus claire que la notation binaire. Effectivement , mieux vaut travailler avec "FFFF" qu'avec "1111111111111111" !!

Notons , pour terminer , que le "haut" de la mémoire , précédemment désigné par "65535" , est également représenté par "FFFF" .

1.6 Le MICRO-PROCESSEUR .

Le NSC 800 de votre CANON X-07 constitue véritablement le "Dieu tout-puissant" . C'est lui qui ordonne , organise et planifie toutes les opérations effectuées à chaque instant au sein de votre machine .

Au niveau "HARD" , il est formé d'un minuscule circuit intégré rassemblant plusieurs milliers de transistors sur une pastille de **SILICIUM** .

Il faut savoir que la rapidité et la puissance d'un ordinateur dépendent principalement du processeur employé . Le NSC 800 est **un micro-processeur 8 bits relativement obsolète** : il est donc tout à fait normal qu'il soit dépassé du point de vue vitesse de traitement , par exemple . En effet , de nouveaux micro-processeurs 16/32 bits équipant la nouvelle génération d'ordinateurs domestiques , se révèlent des plus puissants et des plus véloces (le MOTOROLA 68000 , entre autres ...) . Pour pallier cette "vieillesse" , des **co-processeurs** (un T6834 pour le X-07) déchargent le processeur principal d'une partie des taches pour gagner en vitesse (Voir la deuxième partie de cet ouvrage) .

Gageons que la société CANON nous réserve de bonnes surprises pour l'année 1986 ...

Enfin , chaque type de micro-processeur possède son architecture propre . Nous allons étudier en détail la structure du NSC 800 .

1.7 ARCHITECTURE du NSC 800 .

1.7.1 Les REGISTRES .

Les registres sont constitués par **des mémoires rapides** qui servent soit de "cahier de brouillon" au micro-processeur , soit de repères .

Il en existe **13** plus **8 autres** appelés "registres secondaires" . Certains sont constitués de 8 bits , d'autres de 16 bits (N'oublions pas qu'il existe 16 fils reliant le NSC 800 à la mémoire) et d'autres peuvent être "accolés" pour former des registres de 16 bits (ou "doubles registres") .

Pour mieux appréhender cette importante notion , vous pouvez vous référez à la figure 5 .

Le registre **A** , appelé aussi **"l'ACCUMULATEUR"** , constitue **le registre le plus important** . Beaucoup d'opérations transitent par cette mémoire et , de plus , il garde la majeure partie du temps le résultat des opérations . Vous en appréhenderez l'importance bientôt .

Le registre **F** est un peu spécial car c'est **le registre des drapeaux** (en anglais , "drapeau" se traduit par "FLAG" d'où le registre F) . On y trouve des indications sur le résultat ou le type d'une opération . Ce registre sera passé "au peigne fin" au paragraphe 1.7.3 .

Les registres **B , C , D , E , H et L** sont formés de 8 bits . Ils peuvent fort bien constituer des registres de 16 bits dont les noms seront **BC , DE et HL** . On pourra ainsi travailler directement sur des mots de 16 bits , ce qui est beaucoup plus rapide que de répéter deux fois la même opération sur deux mots de 8 bits .

Les deux registres **IX et IY** , formés de 16 bits , se nomment **"registres d'adressage indexé"** . En fait , ils se révèlent d'une grande utilité bien que d'un **usage relativement restreint** . Pour les utiliser , il faut y stocker une valeur de base à laquelle on ajoute un index (nombre de 8 bits) afin d'obtenir la valeur désirée . Ne paniquez pas , nous expliciterons ceci au chapitre 3.6 .

Le registre **SP** est lui aussi un registre 16 bits . Pour expliquer son utilité , il nous faut d'abord exposer la notion de **"PILE"** . Lorsque le NSC 800 exécute un programme , il peut avoir besoin de sauvegarder des données . Par exemple, s'il doit exécuter un sous-programme puis retourner au programme principal , il rangera à un endroit précis les données du programme principal , exécutera le sous-programme , récupérera les données précédemment sauvées puis reviendra au programme principal .

En fait , ranger les données s'appelle **"EMPILER"** et les retrouver **"DEPILER"** . La pile est donc une zone de la mémoire où le NSC 800 ira ranger les données à sauvegarder "l'une au dessus de l'autre" .

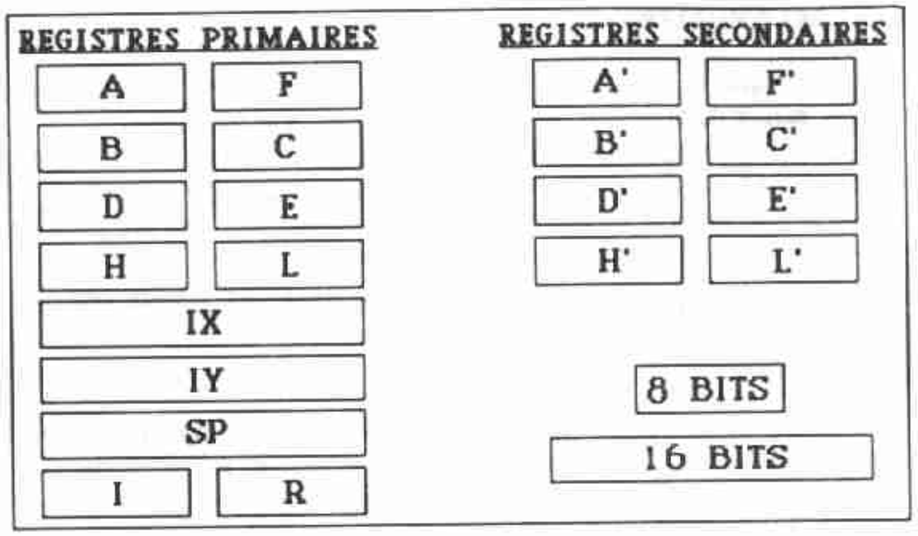


FIGURE 5 : SCHEMA des REGISTRES

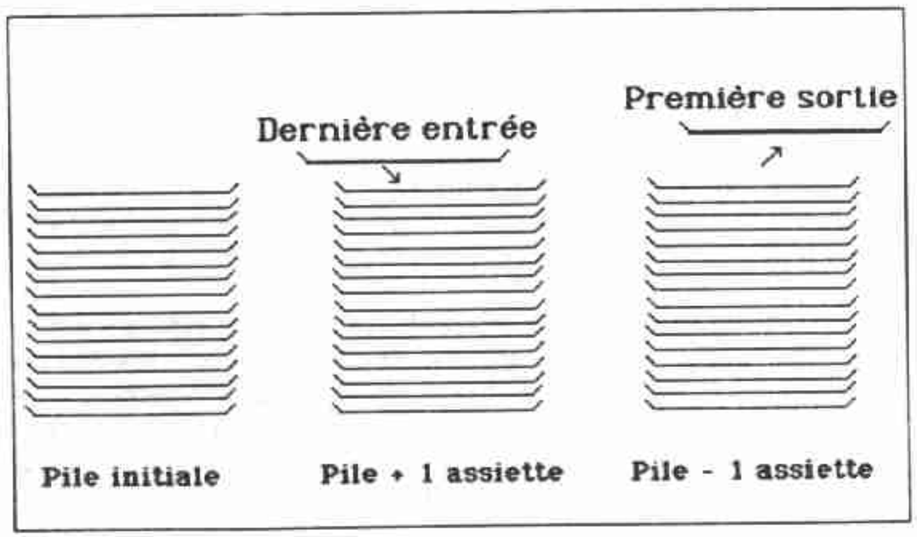


FIGURE 6 : LA PILE

L'emplacement de la dernière donnée empilée est continuellement inscrit dans le registre SP . On dit alors que SP représente **le pointeur de pile** (Stack Pointer en anglais) car **il pointe sur la dernière donnée entrée** . Le fait de conserver l'emplacement (le numéro de la case mémoire , en fait ...) de cette dernière donnée fait que c'est la seule donnée directement accessible . La pile est alors nommée **pile "L.I.F.O."** , de l'anglais "Last In First Out" (en français , premier entré , dernier sorti) . On peut d'ailleurs la comparer à une pile d'assiette (comme dessiné à la figure 6) , les assiettes représentant les données empilées .

I est un registre 8 bits utilisé au sein d'un mode particulier du NSC 800 , **mode lié aux "interruptions"** . Il faut savoir qu'une interruption est constituée par un **événement extérieur** influant sur le déroulement des opérations internes . Par exemple , l'imprimante X-710 envoyant des codes de controle au X-07 représente une interruption . La fonction du registre I est de contenir **une partie d'une adresse vers laquelle se branchera le programme** . Ceci arrivera uniquement en cas d'interruption due à un périphérique . En général , ce registre est assez peu utilisé en programmation .

Le registre **R** n'est pour nous d'aucune utilité . Sa fonction est très technique : il sert à "rafraichir" la mémoire du NSC 800 . En effet , la mémoire ne garde pas éternellement les données qu'elle contient et , de temps à autre , le micro-processeur va les lire pour ensuite les réécrire . Cette opération précise se déroule **à notre insu** et nous est donc totalement étrangère .

Avant de terminer cet exposé sur les registres , il nous faut vous entretenir d'un dernier registre : **PC** . En effet , ce "Program Counter" ou **compteur ordinal** n'est pas accessible en programmation . C'est un registre indiquant l'instruction que va exécuter le NSC 800 . On peut donc comparer son contenu au numéro de ligne du langage BASIC . Sa fonction est d'indiquer le numéro de la case dans laquelle se trouve l'instruction à exécuter . Ce numéro de case se nomme **une adresse** (comme vous vous en doutiez surement ...) .

Pour finir , nous devons préciser une notion primordiale . Nous savons que le NSC 800 ne comprend que des suites de 1 et de 0 . En fait , rien ne distingue dans la mémoire , une donnée d'une instruction . Cette différence à effectuer provient indirectement du registre PC . Effectivement , à la mise sous tension , PC contient **"0000"** et , de ce fait , le NSC 800 exécutera ce qui se trouve à l'adresse "0000" (Ceci constitue un besoin physique ...) . Vous entrevoyez le "Hic" ? ... Eh oui !! Si en "0000" se trouve une instruction , il n'y aura pas de problème ... Par contre , si vous y avez malencontreusement placé **des données** , le NSC 800 se comportera avec elles comme s'il s'agissait d'ordres et les effectuera !! Autant dire qu'il **déraillera complètement** et que le **"RESET"** (placé à l'arrière du X-07) sera le bienvenu !!

Donc c'est à vous de placer correctement données et instructions en mémoire ... Vous êtes livré à vous-même et la moindre erreur sera fatale à vos programmes !

1.7.2 L' UAL , l' UCC ...

Afin d'exécuter les instructions , le micro-processeur possède une **Unité Aritmétique et Logique** , reliée directement aux registres , qui se charge des calculs . On la désigne par "UAL" ou "ALU" selon que l'on emploie le français ou l'anglais .

De plus , on découvre dans le NSC 800 une **unité de décodage des instructions** représentée par un bloc de logique cablé . En fonction des 1 et des 0 qu'elle rencontre , elle effectuera un certain nombre d'opérations constituant l'instruction à exécuter ..

Le micro-processeur possède aussi une logique baptisée **UCC (Unité de Commande et de Controle)** qui permet d'indiquer aux mémoires si elles doivent , par exemple , fournir une donnée ou bien la conserver (Signal Ecriture/Lecture) . ce bus détecte les "tops" de l'horloge interne qui lui servent à rythmer les opérations et les appels des périphériques .

De plus , le NSC 800 a encore un dernier registre de 16 bits permettant d'augmenter ou de diminuer le registre PC en fonction des instructions rencontrées . Afin de vous faire une idée synoptique du micro-processeur , vous pouvez vous référez à la figure 7 .

1.7.3 Les INDICATEURS .

Nous avons vu précedemment que le registre F était différent des autres . En effet , F ne contient pas un mot de 8 bits car chacun de ses bits indique **un état particulier après une opération** . Pour "survoler" sa structure , référez-vous à la figure 8 .

Le bit 0 nommé "CARRY" (Report ou Retenue en français) indique qu'il existe **une retenue** . De plus , nous verrons dans le chapitre "Instructions du NSC 800" qu'il nous permet de tenter des comparaisons et bien d'autres choses encore ...

Le bit 1 nommé "N" (Négatif) sera positionné à 1 si la dernière opération était **une décrémentation ou une soustraction** .

Le bit 2 nommé "P/V" (Parité/oVerflow = débordement en français) indique deux choses : **la parité** (le nombre de bits mis à 1 est pair ou impair) et **le débordement** lors d'une instruction arithmétique (Voir figure 9) .

Les bits 3 et 5 se sont donnés le mot : ils ne servent strictement à **rien!**

Le bit 4 nommé "H" (Half-carry) constitue **une demi-retenue** . Il est égal à 1 lorsqu'il existe une retenue sur les quatre premiers bits d'un opérande (Voir figure 10) . Il est très peu utilisé par les programmeurs .

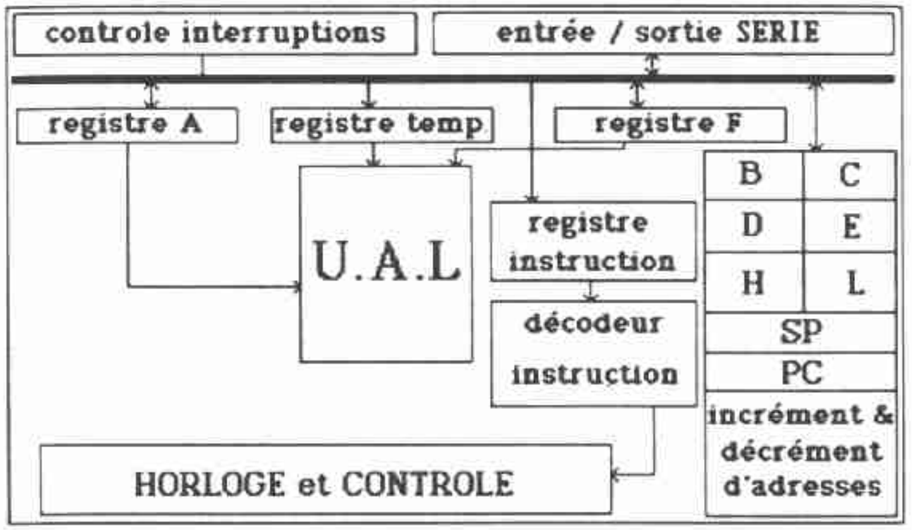


FIGURE 7 : STRUCTURE INTERNE

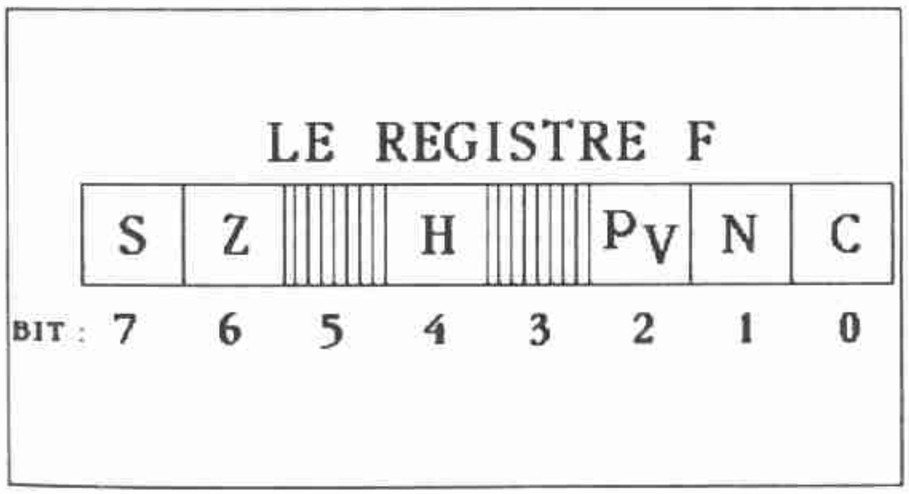


FIGURE 8 : LES INDICATEURS

Le bit 6 nommé "Z" (Zéro) indique si le résultat d'une opération est **égal à 0** .

Le bit 7 nommé "S" (Signe) est affecté (sa valeur change ...) par certaines instructions . Il indique **le signe de la valeur testée** : si S=1 , la valeur est négative ; si S=0 , la valeur est positive .

1.8 POURQUOI UTILISER L' ASSEMBLEUR ?

Après toutes ces généralités importantes à assimiler , vous vous posez peut-être la rituelle question de savoir où tout cela va vous amener .

En effet , c'est bien beau l'ASSEMBLEUR mais , en fait , le BASIC se révèle moins compliqué à apprendre ... Alors pourquoi s'embêter ?

Eh bien , **trois raisons** primordiales vont vous faire réfléchir et vous permettront de juger de l'impact de l'ASSEMBLEUR sur votre programmation .

La première raison réside dans **la rapidité phénoménale** de l'ASSEMBLEUR par rapport au BASIC . En effet , l'ASSEMBLEUR est à peu près **100 fois plus véloce** que le BASIC classique !! Avouez que cela donne envie de réfléchir ...

De plus , l'ASSEMBLEUR occupe en mémoire **beaucoup moins d'octets** que le BASIC : c'est un langage **très compact** . Suivant la nature du programme , un logiciel écrit en ASSEMBLEUR peut occuper **de 2 à 10 fois** moins de place qu'écrit en BASIC !

Enfin , beaucoup de choses supposées "impossibles" à réaliser en BASIC sont désormais offertes sur un plateau avec l'ASSEMBLEUR . En effet , le BASIC ne peut accéder au sous-processeur , aux routines ROM très puissantes ... Le langage machine procure un sentiment d'évasion car **rien n'est impossible à réaliser** . Absolument toutes les possibilités de votre ordinateur vous sont **accessibles** !

Après ce plaidoyer sur l'ASSEMBLEUR , nous pouvons vous assurer qu'il constitue un langage extrêmement puissant , certes difficile à apprendre et à utiliser , mais ô combien fascinant à découvrir ... Et surtout , ne vous arrêtez pas aux premières difficultés car de très bonnes surprises vous attendent !

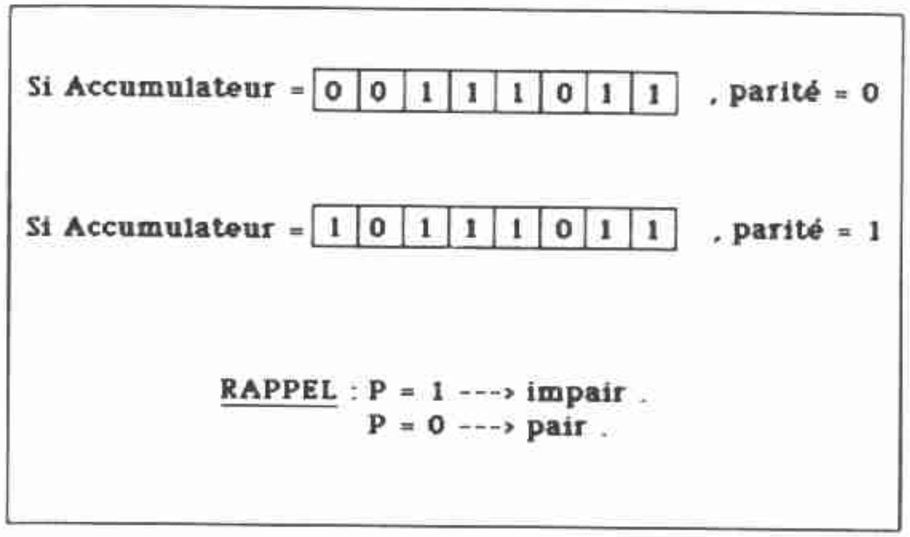


FIGURE 9 : LE BIT N°2

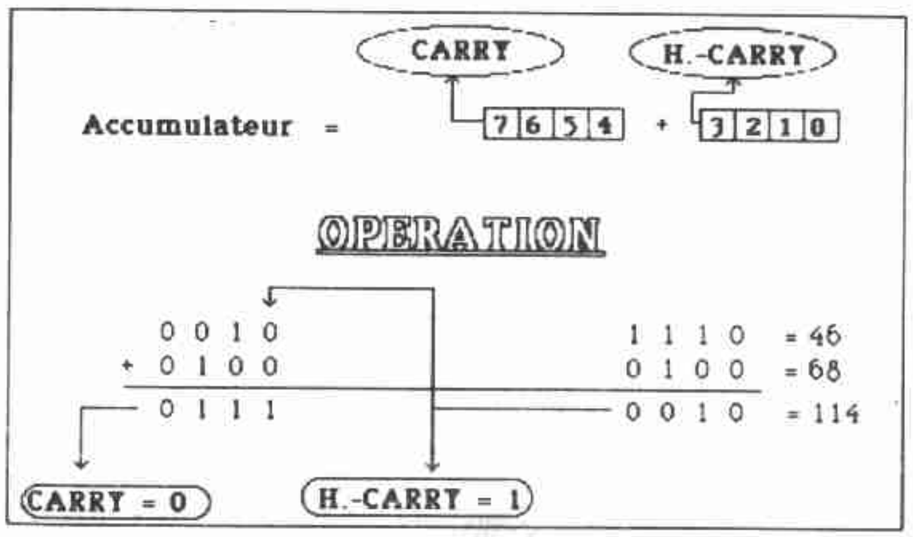


FIGURE 10 : LE BIT N°4

MNEMONIQUE & ASSEMBLEUR

Nous allons attaquer ensemble une des parties les plus importantes du livre . En effet , vous brulez surement de savoir comment développer la programmation de l'ASSEMBLEUR . Tout ceci va être élucidé au cours de ce chapitre ...

2.1 INTRODUCTION .

Nous avons vu précédemment que rien ne différenciait dans la mémoire une donnée d'une instruction . En effet , elles sont toutes deux codées de la même manière , une suite de 1 et de 0 . Si nous devions programmer de cette façon , c'est à dire écrire une longue suite de 1 et de 0 , nous commettrions probablement beaucoup d'erreurs ! De plus , la lecture d'un programme déjà écrit ne serait pas du tout évidente .

Pour pallier à cet état de faits , les constructeurs ont donné un nom à chaque commande . Ce nom constitue **la MNEMONIQUE de l'instruction** . Ainsi , le "GOTO" du BASIC se nommera , en mnémonique Z-80 , **"JP"** (Diminutif de "JumP" qui se traduit par "sauter" en français) . Sa véritable écriture en binaire est "11000011" et "C3" en hexadécimal (Remarquons au passage que cela devient déjà beaucoup plus simple en hexadécimal ...) .

2.2 De L' UTILITE d'un ASSEMBLEUR .

Essayons d'écrire en binaire "aller à la case mémoire 4000" . Très simple , n'est-ce pas ? Cela fait : "11000011" (JP) et "0000111110100000" (4000 codé sur 16 bits) . De plus , pour simplifier la chose (!) , il faut écrire les 8 bits les plus importants (dits de **"poids fort"**) à l'adresse de poids fort . En effet , les mots étant stockés sur 8 bits , on obtiendrait logiquement :

Adresse	Contenu	
02	10100000	Seulement , si l'on stocke les nombres de cette
01	00001111	façon , la partie faible de l'adresse du saut sera
00	11000011	stockée à l'adresse la plus forte . Il faut donc inverser la deuxième partie de l'adresse .

En fait , on trouvera dans la mémoire la configuration suivante :

Adresse en DECIMAL	Contenu en BINAIRE	Contenu en HEXA.
02	00001111	0F
01	10100000	A0
00	11000011	C3

Comme vous avez pu le remarquer , toutes ces opérations deviennent très rapidement **fastidieuses** à effectuer "à la main" . Par contre , c'est le travail typique dont peut se charger un ordinateur .

Par conséquent , on a écrit un logiciel auquel on fournit la liste de toutes les mnémoniques , qui les transforme en binaire puis les range à l'adresse désirée . Le listing en mnémoniques s'intitule **"le code source"** , les mots binaires obtenus représentant **"les codes objets"** . Le logiciel se nomme alors un **"ASSEMBLEUR"** .

2.3 Les LABELS .

Un label constitue **une référence placée dans le code source** . On pourra ensuite y faire appel , l'ASSEMBLEUR se chargeant de la transformer .

Ainsi , au lieu de "JP 4000" , nous aurons pu écrire **"JP LABEL"** si nous avions défini auparavant "4000" comme un "LABEL" . Ceci s'appelle , en termes techniques , **l'adressage symbolique** . Ce mode d'adressage n'existe pas dans le BASIC du X-07 mais , à titre d'exemple , les micro-ordinateurs de la gamme SHARP (PC 1212 , 1500 ...) peuvent utiliser ce type d'adressage (Ex : GOTO "DEBUT" , GOSUB "FIN" ...) .

Mais les labels **ne sont pas strictement réservés** à l'adressage symbolique . Effectivement , nous pouvons écrire LD A,"SOMME" si nous avons précédemment affecté une valeur au symbole "SOMME" (Ex : "SOMME" = 11 ; LD A,"SOMME" est alors équivalent à LD A,11) . Notons que la commande "LD" charge un nombre dans le registre spécifié : nous verrons ceci plus en détail au chapitre 4 .

En fait , les possibilités réelles d'un ASSEMBLEUR varient et sont toujours exposées dans sa notice .

2.4 FONCTIONNALITES de L' ASSEMBLEUR .

Vous devez probablement vous demander la façon d'introduire le code source en mémoire ... Cette opération se pratique avec **un éditeur de textes** intégré ou non au logiciel ASSEMBLEUR . Parfois , cet éditeur de textes s'identifie à **l'éditeur BASIC** . En effet , les deux ASSEMBLEURS pour X-07 (Voir ANNEXE 2) utilisent l'éditeur BASIC très complet du CANON afin d'éviter d'en réécrire un .

Ensuite , **la phase d'assemblage** proprement dite débute . Elle est divisée , en général , en deux parties appelées **des "PASSES"** .

La première PASSE examine consciencieusement le code source entré en mémoire et détermine la longueur de chaque instruction . En effet , la longueur d'une instruction peut varier de **1 à 4 octets** (ex : RET occupe un octet et JP 4000 en occupe trois ...) . De plus , une table des labels rencontrés est construite et la syntaxe du code source est minutieusement controlée .

La deuxième PASSE permet à l'ASSEMBLEUR de générer le code objet de chaque instruction . De plus , une adresse réelle est affectée à chaque label ou symbole .

Nous allons maintenant étudier le format du code source , équivalent de nos lignes BASIC .

2.5 FORMAT du CODE SOURCE .

Le code source est constitué de lignes écrites les unes à la suite des autres. Chaque ligne possède deux types de formats :

_ **Le format FONCTIONNEL** , traité par l'ASSEMBLEUR , se compose de quatre champs distincts : LABEL , OPERATION , OPERANDE , COMMENTAIRE .

_ **Le format COMMENTAIRE** débute par un caractère précis (; ou ' en général) . Les lignes "commentaires" sont uniquement destinées à documenter le listing source . Elles correspondent aux lignes débutant par un "REM" en BASIC .

Voici un exemple de "lignes source" :

```
' EXEMPLE
FIN     JP      4000    'On saute à la fin du programme
```

Champs : label/opération/opérande/commentaire

La première ligne est une ligne de commentaires et la deuxième est fonctionnelle . Sous chaque instruction , le type du champ est défini . A partir de cela , nous allons étudier les particularités de chaque champ .

2.5.1 Le champ LABEL .

Ce champ débute toujours en première position de la ligne source par un caractère spécial (ex : #) . Il est formé de caractères numériques ou alphabétiques , sans espace . Sa longueur est déterminée par les performances de l'assembleur utilisé . En général , ce champ se termine par le séparateur BASIC ":" .

Ce champ label se révèle **optionnel** à l'usage : on peut très bien s'en passer bien que l'utilisation de labels clarifie le programme .

Exemples de LABELS : #EV , #1 , #DEBUT ....

2.5.2 Le champ OPERATION .

Ce champ commence immédiatement après le champ LABEL si celui-ci est présent . Il est **toujours obligatoire dans le format fonctionnel** et se compose d'une **instruction ou d'une pseudo-instruction** (Voir chapitres 4 et 5) .

Exemples : CALL , LD , JP , RET ; ORG , DEFM ...

2.5.3 Le champ OPERANDE .

Ce champ est pratiquement toujours présent (sauf dans le cas de certaines instructions telles RET , NOP ou HALT ...) et lié au **type d'opération** effectué immédiatement avant .

Ce champ peut comporter un ou deux arguments suivant le cas . Par exemple , RET NZ est un opérande comprenant un argument (NZ) et LD A,B en comprend deux (A et B) .

Chaque argument peut adopter l'une des **significations suivantes** :

_ Une adresse .
_ Un pointeur adresse .
_ Un registre .
_ Un pointeur registre .
_ Une donnée .
_ Un mot-clé de condition .

Ces différents types d'argument vont être détaillés dans le paragraphe 2.6 .

2.5.4 Le champ COMMENTAIRE .

Le champ "commentaire" doit obligatoirement être placé après le champ opérande .

Il est signalé à son début par **un caractère spécial** du style "'" ou ";" . Ce champ est optionnel et permet **une meilleure documentation** du listing source .

2.6 Les ARGUMENTS de L' OPERANDE .

Nous avons vu précédemment que l'argument d'un opérande pouvait prendre plusieurs significations . En effet , l'argument d'opérande peut être constitué par ...

2.6.1 Une ADRESSE .

Il s'agit d'une **adresse mémoire codée sur 16 bits** ne pouvant être contenue que dans un double registre (BC , DE , HL ...) . Par exemple , on peut trouver LD HL,4520h ; LD IX,2020h ... ("h" désigne la base 16) .

2.6.2 Un POINTEUR ADRESSE .

Un pointeur adresse permet d'accéder à **l'octet pointé par une adresse mémoire** . Par exemple , en exécutant LD HL,(2000h) , on ne stocke pas l'adresse 2000h dans le double registre HL mais l'octet situé à l'adresse 2000h . Manière tout à fait indirecte !! Notez que **les parenthèses** spécifient ce type d'argument .

2.6.3 Un REGISTRE .

En effet , l'argument d'opérande prendra alors le nom de l'un **des 16 registres suivants** :**BC , DE , HL , SP , IX , IY , AF , A , B , C , D , E , H , L , I ou R** . Vous ne remarquez rien ? ... Le registre F ne peut être utilisé seul car c'est un registre spécial (Voir le paragraphe 1.7.3) . Par exemple , on peut avoir OR D ; POP IX ; LD B,E ...

2.6.4 Un POINTEUR REGISTRE .

Nous avons vu au paragraphe 2.6.2 une façon indirecte de désigner un octet via une adresse . Le fait de prendre un registre au lieu d'une adresse ne change rien à l'affaire !

En effet , ce pointeur est constitué par **l'un des registres suivants** : **(DE) , (HL) , (BC) , (IY) , (IX) et (SP)** . Notez au passage que ces registres sont pourvus de parenthèses comme pour le pointeur adresse .

Je pense que vous avez deviné l'action de ce pointeur . Il permet d'accéder à l'octet pointé par l'adresse contenue dans le registre mentionné entre les parenthèses ... OUF !!!

Par exemple , si le registre HL contient l'adresse 4500h et que l'on exécute LD B,(HL) , qu'arrivera t-il ? ... Eh bien , le registre B contiendra l'octet situé en 4500h .

2.6.5 Une DONNEE .

Il peut s'agir d'une donnée constituée de **8** ou de **16 bits** selon le type d'instruction utilisée . Cette donnée peut adopter plusieurs formes différentes exposées au paragraphe 2.7 .

Quelques exemples : LD BC,4543h ; LD B,4 ; LD A,18h ...

2.6.6 Un MOT-CLE de CONDITION .

Les instructions du NSC 800 peuvent tester la présence de conditions particulières matérialisées par les fameux indicateurs du registre F (Voir paragraphe 1.7.3) .

Un mot-clé correspond à chacune de ces conditions . Il se trouve toujours en premier argument de l'opérande des instructions de branchement (Ex : JP , CALL , RET ...) .

Il existe **8 mots-clés** différents :

Z	-----›	Zéro
NZ	-----›	Non Zéro
C	-----›	Carry
NC	-----›	Non Carry
PE	-----›	Parité Paire
PO	-----›	Parité Impaire
M	-----›	Moins (négatif)
P	-----›	Plus (positif)

Voici <u>deux exemples</u> afin d'éclaircir vos idées :

1/
```
CALL NZ , 4000h
LD HL , (1000h)
```

Ici , si l'indicateur NZ est positionné à 1 , le programme se poursuivra en exécutant le sous programme situé à l'adresse 4000h puis reviendra . Sinon , si NZ n'est pas positionné , le programme n'ira pas en 4000h et chargera seulement , dans le registre HL , l'octet situé en 1000h .

2/
```
RET C
JP 5000h
```

Ici , si l'indicateur se trouve être égal à 1 , le programme exécutera le RET (RETURN) . Sinon , il poursuivra à l'adresse 5000h à cause du JP (JUMP) .

Après avoir terminé le détail des champs , occupons-nous des données ...

2.7 DEFINITION des DONNEES .

Une donnée en ASSEMBLEUR peut être constituée par **une constante ou une adresse mémoire** . On peut définir une donnée ...

2.7.1 En DECIMAL .

La notation décimale est la plus répandue avec l'hexadécimal . Les données peuvent être signées ou non (Elles peuvent posséder un signe négatif ou positif) .

Exemples : LD BC,6000 ; LD A,32 ; LD C,-9 ...

2.7.2 En HEXADECIMAL .

Suivant l'ASSEMBLEUR utilisé , un signe sera placé devant ou derrière la donnée pour préciser qu'il s'agit d'un nombre hexadécimal . Les symboles peuvent être : $, h , H , &H , & ...

Exemples : LD IX,22h ; LD A,29h ...

2.7.3 En BINAIRE .

Cette notation peu commode (comme nous l'avons déjà remarqué ...) s'utilise assez rarement et tous les ASSEMBLEURS ne l'acceptent pas .

Exemples : LD HL,11010001B ; LD SP,11000011B ...

Notez que la lettre "B" spécifie la base du nombre .

2.7.4 Sous forme d'un CARACTERE ASCII .

Ecrire un nombre sous forme ASCII est très ingénieux car la lecture du listing source en est grandement facilitée . On encadre en général le caractère entre deux guillemets .

Exemples : LD B,"A" (équivaut à LD B,65) .

2.7.5 Sous forme d'un LABEL ou d'un SYMBOLE .

Si vous avez la précaution de définir des labels , vous pouvez définir des données "parlantes" .

Exemples : LD A,"VALEUR1" ou JP "FIN" ...

2.7.6 Sous forme d'une EXPRESSION .

Certains ASSEMBLEURS permettent ce genre de désignation . En effet , vous pouvez définir une donnée comme le résultat d'un expression à calculer .

Exemples : LD BC,2*(10-7)+5 AND 17 ; JP DEBUT + 8 ...

2.7.7 Sous forme OCTALE .

Enfin , la notation octale , très peu utilisée , peut servir à définir une donnée . En effet , la base 8 était très utilisée sur les premiers ordinateurs . Très peu de matériels acceptent dorénavant cette notation .

Exemples (désuets !!) : LD B,640 ; LD A,1010 ...

Notez la lettre "O" désignant la base 8 .

Ce chapitre sur la structure intime de l'ASSEMBLEUR s'achève . Vous pouvez vous référer dès maintenant aux programmes du chapitre 6 pour revoir en détail toutes les concepts exposés précédemment .

Après cela , vous pourrez continuer votre quête du savoir en notre compagnie et celle des modes d'adressage ...

LES DIVERS MODES D' ADRESSAGE DU NSC 800

Les modes d'adressage représentent **les différents moyens d'accéder à une donnée** . Ils facilitent considérablement l'écriture d'un logiciel tout en augmentant sa vitesse d'exécution . Le NSC 800 en possède **sept** explicités ci-dessous .

3.1 L' ADRESSAGE IMMEDIAT .

Il constitue l'adressage le plus simple que vous puissiez rencontrer . En effet , il correspond à l'exemple précédemment évoqué "JP 4000" . Il convient uniquement de fournir l'adresse après la commande . Ainsi , charger le registre A avec le chiffre 9 s'écrit "LD A,9" (LD signifiant "LoaD" , c'est à dire charger en français) .

3.2 L' ADRESSAGE REGISTRE .

On part du principe que l'opérande se trouve dans un des registres du NSC 800 . On obtiendra ainsi "LD A,B" (charge le registre A avec le contenu du registre B) mais aussi "JP HL" (saute à l'adresse contenue dans le registre HL) .

3.3 L' ADRESSAGE INDIRECT REGISTRE .

On utilise un registre afin de pointer sur une adresse mémoire . Par exemple , l'instruction "LD A , (HL)" , avec HL contenant la valeur 4000h , chargera dans l'accumulateur la valeur se situant à l'adresse 4000h . Notez que les parenthèses marquent cette indirection .

3.4 L' ADRESSAGE DIRECT .

Il fonctionne de la même manière que l'adressage indirect registre . Mais , au lieu de placer l'adresse dans un registre , on la donne directement . Ainsi , "LD A , (5000)" chargera dans le registre A la valeur située à l'adresse 5000 .

3.5 L' ADRESSAGE RELATIF .

Cet adressage n'est utilisé que pour les sauts . Nous avons vu précédemment l'instruction de saut "JP" . Parallélement , il en existe une autre notée "JR" (Jump Relative , en anglais) représentant le saut relatif .

Après l'instruction "JP" , nous devons fournir l'adresse où nous désirons parvenir . Avec l'instruction "JR" , nous fournissons le nombre de cases mémoires dont nous voulons nous déplacer aussi bien en avant qu'en arrière .

Ce déplacement est fourni sur un octet . De ce fait , cela entraine une restriction importante puisque avec un octet , nous ne pouvons coder que 256 cases . Le déplacement ne pourra donc dépasser **127 cases en avant et 128 cases en arrière** (Voir figure 11) .

3.6 L' ADRESSAGE INDEXE .

Il s'agit en fait de s'adresser à une case mémoire par un déplacement en avant ou en arrière . Cet adressage se pratique par rapport à une adresse précise contenue dans l'un des deux registres d'indexage IX ou IY (Voir paragraphe 1.7.1) .

Ainsi , nous pouvons écrire "LD A , (IX + 5)" : le registre A sera chargé avec l'octet situé à l'adresse stockée dans le registre IX augmentée de 5 .

Comme précédemment , le déplacement est limité à 127 en avant et 128 en arrière (Voir figure 12) .

3.7 L' ADRESSAGE BIT .

Maintenant , on ne s'adresse plus à un octet mais à un bit situé dans un registre ou dans une case mémoire .

Ainsi , la commande "SET 2 , A" positionne à 1 le bit 2 du registre A , sans modifier les autres bits . On peut tester un bit ou le mettre à 0 de la même façon .

A la fin de ce chapitre , vous devez connaitre à peu près la moitié de la structure de l'ASSEMBLEUR . Il ne reste plus qu'une étape , étape ô combien primordiale : **les instructions du NSC 800** .

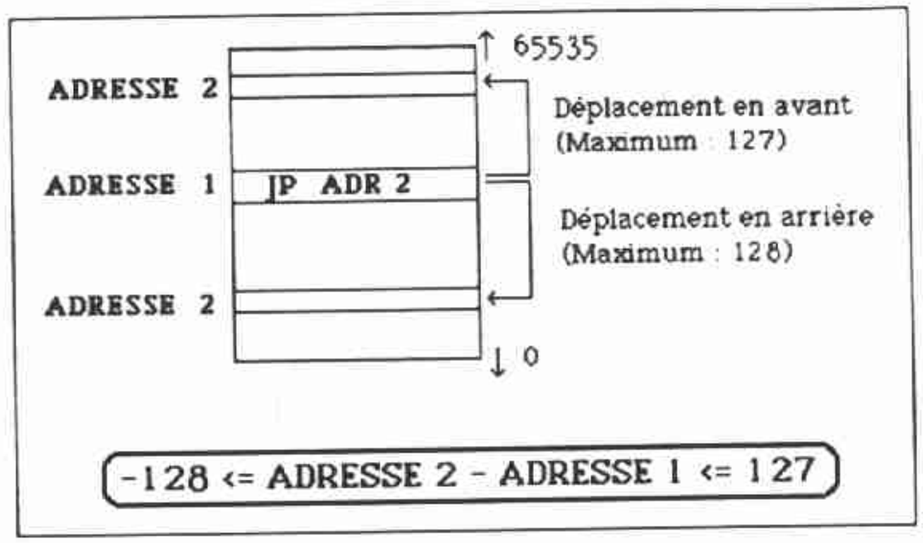


FIGURE 11 : L' ADRESSAGE RELATIF

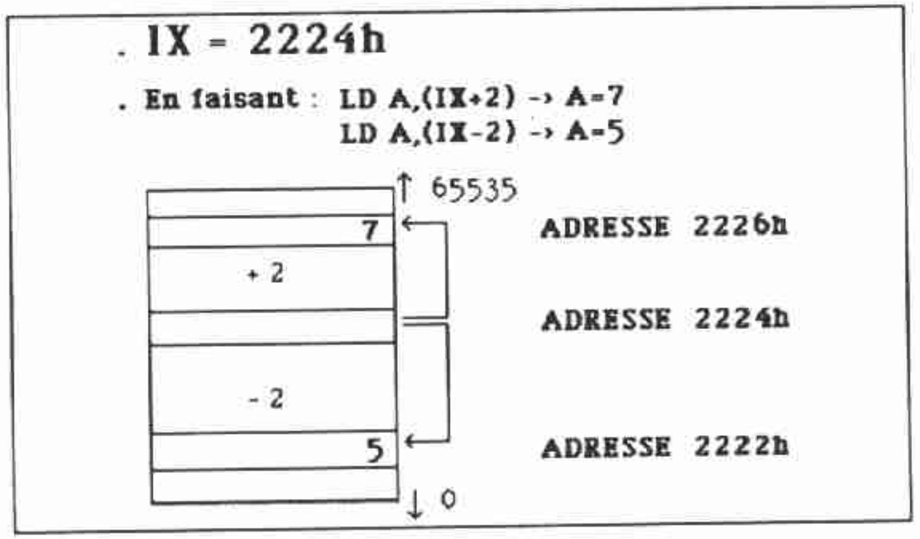


FIGURE 12 : L' ADRESSAGE INDEXE

LES INSTRUCTIONS DU NSC 800

Nous allons maintenant aborder le "gros morceau" de cette première partie consacrée à l'assembleur Z-80 . En effet , nous allons vous exposer toutes les instructions disponibles , soit quelques **696 commandes distinctes** !!

Rassurez-vous , nous n'allons tout de même pas vous les donner en vrac ... Nous avons distingué **11 familles d'opérations différentes** , décrites une à une dans les pages suivantes .

De plus , un tableau général , situé à l'annexe 1 , récapitule toutes les instructions disponibles . N'hésitez surtout pas à vous y reporter en cours de lecture ...

Avant de commencer , voici la liste des symboles utilisés :

nn	désignera une adresse sur 16 bits
v	désignera une valeur sur 8 bits
vv	désignera une valeur sur 16 bits
dpl	désignera un déplacement dans l'opérande des instructions en mode relatif
r	désignera un registre 8 bits
rr	désignera un registre 16 bits

4.1 CHARGEMENT et RANGEMENT .

On utilise l'instruction **LD** (LoaD) . Les informations transiteront vers la mémoire d'un registre ou vice-versa .

Ainsi , **LD A,(nn)** charge l'accumulateur avec la valeur contenue dans la . case mémoire d'adresse nn . On a aussi :

LD A,(HL)	**et inversement**	**LD (HL),A**
LD B,(HL)		**LD (HL),B**
LD r,(HL)		**LD (HL),r**

Ceci est donc valable pour tous les registres . Ici , c'est le registre HL qui sert de pointeur mémoire . De plus , le regitre A possède un privilège :

LD A,(BC)	**et**	**LD (BC),A**
LD A,(DE)		**LD (DE),A**

Attention ! Seul le registre A peut accéder à la mémoire en se servant des registres BC et DE comme pointeurs . Nous avons aussi les adressages indirects indexés :

LD (IX + dpl),r et **LD r,(IX + dpl)**
LD (IY + dpl),r **LD r,(IY + dpl)**

En ce qui concerne LD (IX + dpl),r (respectivement LD (IY + dpl),r) , le registre considéré est chargé à l'adresse représenté par IX + dpl (respectivement IY + dpl) . Les instructions LD r,(IX + dpl) et LD r,(IY + dpl) permettent de charger le registre r avec le contenu de l'adresse IX + dpl ou IY + dpl . De plus , on a les commandes suivantes :

LD (HL),v
LD (IX + dpl),v
LD (IY + dpl),v

Dans ce cas , on range directement la valeur v en mémoire à l'adresse correspondante (adresse contenue dans le registre HL , IX + dpl ou IY + dpl) . Par exemple , LD (HL),8 , avec HL contenant 4000 , va stocker à l'adresse 4000 le chiffre 8 . Voyons maintenant l'adressage direct 16 bits :

LD (nn),rr et **LD rr,(nn)**

Le contenu du double registre (BC , DE ou HL) est directement rangé ou chargé à l'adresse nn (Poids faible à l'adresse faible nn et poids fort à l'adresse forte nn+1) . Par exemple , LD (7000),BC charge le contenu du double registre BC à l'adresse 7000 . De même , LD HL,(9000) charge le contenu de l'adresse 9000 dans le double registre HL .

Pour terminer cette rubrique , voici deux instructions très particulières : **LDD et LDI** . L'octet dont l'adresse est pointée par le registre HL est chargé à l'adresse pointée par le registre DE . Simultanément , le registre BC se trouve décrémenté de 1 et provoque le passage à 0 de l'indicateur P/V lorsqu'il atteint une valeur nulle . De plus , les registres DE et HL sont décrémentés par LDD et incrémentés par LDI .

4.2 CHARGEMENT des REGISTRES et ECHANGE .

L'instruction **LD r,n** permet de charger le registre r avec la valeur n . L'instruction **LD rr,nn** permet de charger le registre rr avec la valeur nn . De plus , **LD r_1,r_2** charge le registre r_1 avec la valeur du registre r_2 : aucune restriction !! On peut très bien écrire **LD A,A** même si ce n'est pas très utile ...

Il existe trois autres instructions qui sont : **LD SP,HL**
LD SP,IX
LD SP,IY

Le registre SP (Stack Pointer) est chargé avec la valeur d'un de ces trois registres .

Plus précisément , voyons maintenant l'instruction **EXX** . Nous avons vu précédemment un deuxième groupe de registres (A' à L') : EXX échange les deux familles de registres , les mémoires A' à L' se comportant alors exactement comme les registres A à L . Par exemple , **EX AF,AF'** échange le registre AF avec le registre AF' . Il existe aussi **EX (SP),HL , EX (SP),IX et EX (SP),IY** : l'échange est effectué entre l'un des doubles registres HL , IX et IY d'une part , et la case mémoire pointée par le registre SP d'autre part (la pile ...) .

4.3 FONCTIONS ARITHMETIQUES .

4.3.1 ADDITION avec et sans CARRY .

Les deux instructions **ADD A,v** et **ADC A,v** permettent d'ajouter la valeur v à celle contenue dans le registre A . L'accumulateur contiendra donc le résultat en fin de calcul . Dans le cas de la commande ADC , le bit CARRY est positionné s'il existe une retenue .

De la même façon , **ADD A,r** et **ADC A,r** permettent d'ajouter le contenu du registre r à celui du registre A qui contiendra le résultat final . D'autre part, les instructions suivantes permettent quantité de choses :

ADD A,(HL) **ADC A,(HL)**
ADD A,(IX + dpl) **ADC A,(IX + dpl)**
ADD A,(IY + dpl) **ADC A,(IY + dpl)**

Ici , le contenu de la case mémoire adressée par le registre HL (ou par IX + dpl , IY + dpl) est ajouté au contenu de l'accumulateur .

De plus , **ADD HL,rr** et **ADC HL,rr** (rr pouvant représenter les registres HL , DE , BC ou SP) permettent d'ajouter le contenu du registre rr au registre HL jouant le role d'un accumulateur .

Enfin , **ADD IX,rr** et **ADD IY,rr** (avec rr pouvant représenter BC , DE , HL ou SP) permettent d'effectuer l'addition du contenu du registre rr avec le contenu du registre IX ou IY . On remarque qu'il n'existe pas les instructions **ADD IX,rr** ou **ADC IY,rr** .

4.3.2 SOUSTRACTION avec et sans CARRY .

Nous gardons le même principe que pour l'addition à part une petite **exception de syntaxe** . En effet , nous avons écrit **ADD A,r** mais la soustraction sans CARRY se trouve **privée de son premier argument** : elle s'écrira **SUB r** , le premier argument étant toujours constitué par **le registre A** . Nous aurons donc deux instructions notées **SUB v** et **SBC A,v** : elles soustraient la valeur v au contenu de l'accumulateur avec , dans le cas de SBC A,v , la valeur du CARRY otée du résultat .

De plus , **SUB r** et **SBC A,r** permettent de soustraire le contenu du registre r au contenu de l'accumulateur . Attention !! Si **SUB A** donne toujours 0 , la commande **SBC A,A** peut donner 0 ou FF (-1 sur le CANON X-07) si le CARRY était égal à 1 avant l'opération .

De plus , il existe aussi les instructions suivantes :

SUB (HL)	**SBC A,(HL)**
SUB (IX + dpl)	**SBC A,(IX + dpl)**
SUB (IY + dpl)	**SBC A,(IY + dpl)**

Ici , on soustrait à l'accumulateur le contenu de la case mémoire pointée par l'argument .

Enfin , on a : **SBC HL,BC** **SBC HL,DE**
SBC HL,HL **SBC HL,SP**

Il n'existe pas de SUB équivalent . L'accumulateur est représenté par le registre HL .

4.3.3 INCREMENT et DECREMENT .

Ce n'est pas autre chose qu'une addition ou une soustraction d'une unité . Ces instructions sont très utilisées en ASSEMBLEUR et correspondent à **INC r** et **DEC r** . Un exemple : LD A,45
DEC A
Après l'instruction DEC A , le registre A contiendra la valeur 44 .

Le principe est le même pour les instructions suivantes :

INC (HL)	**DEC (HL)**
INC (IX + dpl)	**DEC (IX + dpl)**
INC (IY + dpl)	**DEC (IY + dpl)**

Ici , l'octet contenu dans la case mémoire pointée par le registre en argument se trouve incrémenté ou décrémenté . De plus , sur 16 bits , on découvre les instructions **INC rr** et **DEC rr** (avec rr pouvant représenter les registres BC , DE , HL , IX , IY ou SP) .

Attention !! Il existe cependant un piège malgré l'apparente désinvolture de ces instructions :

```
LD HL,FFFF
INC HL        donnera ? ... Eh bien , le registre HL contiendra 0000 et
                            aucun indicateur ne sera positionné !!!
```

Une dernière chose : il existe un mode d'adressage particulier appelé **"implicite"** . L'instruction sait automatiquement à quel registre elle doit s'adresser ... Cette instruction se nomme **DAA** et effectue **l'ajustement décimal du contenu de l'accumulateur** . Voici un petit exemple :

```
LD A,8      L'accumulateur contiendra la valeur "0E" avant l'instruction
LD B,6      DAA . Puis , la valeur 14 sera stockée dans le registre A . La
ADD A,B     commande DAA ajoute 6 au contenu de A si la valeur des 4
DAA         bits de poids faible dépasse 9 .
```

C'est une manière de transformer **le binaire en décimal** .

4.4 Les FONCTIONS LOGIQUES .

Toutes ces instructions logiques opèrent implicitement sur le registre A , celui-ci ne figurant pas dans l'opérande .

4.4.1 L'instruction AND .

Tout d'abord , traçons la table de vérité de cette fonction . Rappelons qu'une table de vérité donne la valeur de l'assertion résultante en fonction des assertions que l'opérateur fait intervenir .

Pour la fonction AND , on obtient :

A	B	A AND B
0	0	1
0	1	0
1	0	0
1	1	1

L'instruction **AND v** effectue le AND logique de l'accumulateur avec la valeur v . Avec les fonctions **AND (HL)** , **AND (IX + dpl)** , **AND (IX + dpl)** , le AND est exécuté avec l'octet pointé par l'opérande .

L'instruction **AND r** effectue le AND avec un registre . De plus , la commande **AND A** peut déterminer la valeur qui se trouve dans le registre A . Ainsi , si nous avons LD A,(HL) , AND A positionnera les indicateurs en fonction du contenu de l 'accumulateur .

4.4.2 L'instruction OR .

Ecrivons la table de vérité de cette fonction :

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

On possède les mêmes possibilités qu'avec le AND logique :

OR v
OR (HL)
OR r
OR (IX + dpl)
OR (IY + dpl)

Notons au passage que **OR A** fait aussi "ressortir" les indicateurs Z et S .

4.4.3 La disjonction logique ou XOR .

Voici la table de vérité de XOR :

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Les possibilités exploitables restent identiques :

XOR v
XOR (HL)
XOR r
XOR (IX + dpl)
XOR (IY + dpl)

A noter : **XOR A** permet de remettre à 0 l'accumulateur en un seul octet. En effet , l'instruction LD A,0 en nécessite deux !! Astucieux ...

4.4.4 COMPLEMENT et NEGATION .

Ces commandes réalisent le complément à 1 (NOT) ou le complément à 2 (Négation) de l'accumulateur . Elles se nomment **CPL** et **NEG** . Voici un exemple vous permettant de saisir le rôle de ces deux instructions :

LD A,14h	Après l'instruction CPL , le registre A contiendra "EB" .
CPL	Après l'instruction NEG , l'accumulateur contiendra la
NEG	valeur "15" en hexadécimal .

4.5 COMPARAISONS .

Bien que les comparaisons et les décalages (Voir paragraphe 4.6) soient des fonctions logiques , nous avons préféré les séparer dans un pur souci de clarté.

Les comparaisons entre deux valeurs ne sont en fait qu'une soustraction provoquant la "montée" du CARRY s'il n'y a pas report du BIT 7 . En fait , inutile de réfléchir autant car il suffit simplement de savoir que :

CARRY = 0 -----> **Accumulateur >= Opérande**

CARRY = 1 -----> **Accumulateur < Opérande**

On note que la comparaison se fait toujours avec le registre A . Nous obtenons les commandes suivantes :

CP v
CP r
CP (HL)
CP (IX + dpl)
CP (IX + dpl)

De plus , il existe les commandes **CPI** et **CPD** (en rapport avec **LDI** et **LDD**) : le registre HL pointe l'octet à comparer et le registre BC sert de compteur .

4.6 LES DECALAGES .

Il existe cinq grands types de décalages étudiés en détail ci-dessous .

4.6.1 DECALAGES CIRCULAIRES gauche et droite .

Les bits de l'opérande sont décalés vers la gauche ou vers la droite . De plus , le bit sortant d'un coté entre de l'autre , tout en étant recopié dans le bit CARRY (Voir figure 13) .

On obtient les possibilités suivantes :

RLC r
RLC (HL)
RLC (IX + dpl)
RLC (IY + dpl)
RLCA
RRC r
RRC (HL)
RRC (IX + dpl)
RRC (IY + dpl)
RRCA

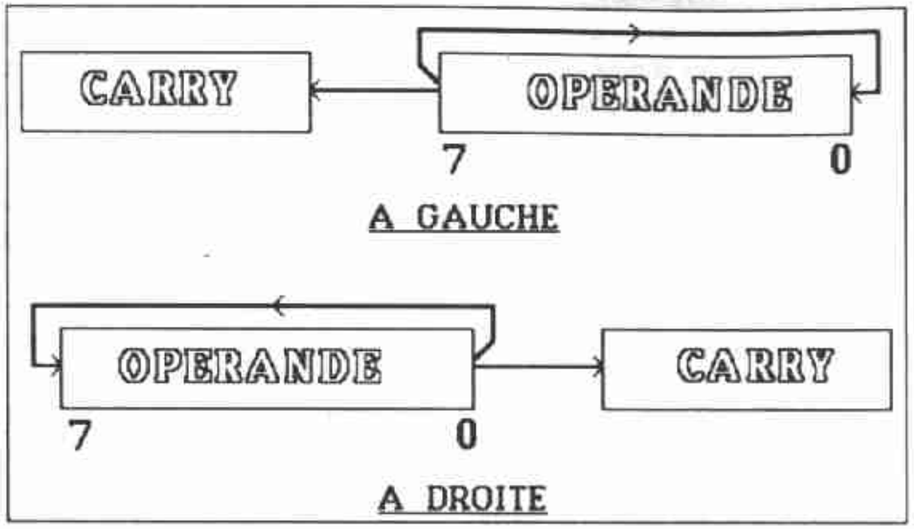


FIGURE 13 : DECALAGES CIRCULAIRES

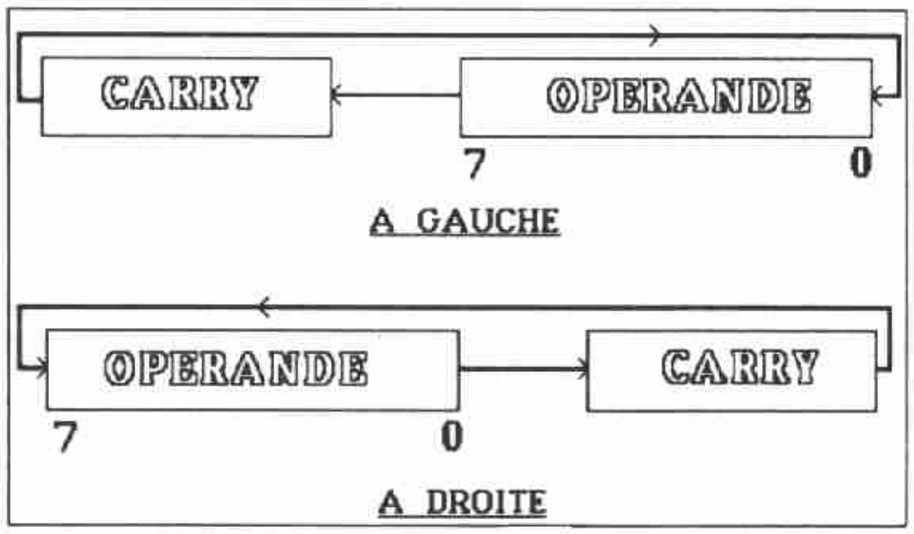


FIGURE 14 : DECALAGES / CARRY

Un peu de langue anglaise : **RLC** signifie **Rotate Left Circular** et **RRC** se traduit par **Rotate Right Circular** .

Les instructions **RLCA** et **RRCA** ne sont pas spécifiques au micro-processeur Z-80 . En effet , elles proviennent du **micro-processeur 8080 de la société INTEL** et fonctionnent de la même manière que les commandes **RLC A** et **RRC A** à part que les indicateurs sont positionnés avec ces dernières .

4.6.2 DECALAGES à travers le CARRY , à droite et à gauche .

Tout se passe comme si l'opérande ne faisait plus 8 bits mais 9 et qu'il soit décalé d'une position binaire . Le bit sortant d'un coté entre dans le bit CARRY et l'ancienne valeur du CARRY entre de l'autre coté de l'opérande (Voir figure 14) . Pour les instructions **RLA** et **RRA** que vous allez découvrir ci-dessous , les remarques précédentes sont valables (issues du 8080 ...) . Voici les commandes disponibles :

RL r	**RR r**
RL (HL)	**RR (HL)**
RL (IX + dpl)	**RR (IX + dpl)**
RL (IY + dpl)	**RR (IY + dpl)**
RLA	**RRA**

4.6.3 DECALAGES OUVERTS ARITHMETIQUES droite et gauche .

Ces décalages sont **"ouverts"** car le bit sortant est perdu alors que le bit entrant est nul . De plus , **le décalage à droite** conserve le signe (Voir figure 15) . Les possibilités restent identiques :

SLA r	**SRA r**
SLA (HL)	**SLA (HL)**
SLA (IX + dpl)	**SRA (IX + dpl)**
SLA (IY + dpl)	**SRA (IY + dpl)**

SLA se traduit en anglais par **Shift Left Arithmetic** et **SRA** par **Shift Right Arithmetic** .

4.6.4 DECALAGES "OUVERTS LOGIQUES" à droite .

Pour "contrebalancer" l'instruction **SRA** qui recopie le bit de signe après le décalage , il existe une autre instruction nommée **SRL** . l'octet mémoire pointé par l'opérande sera décalé logiquement de une position vers la droite (Voir figure 16) .

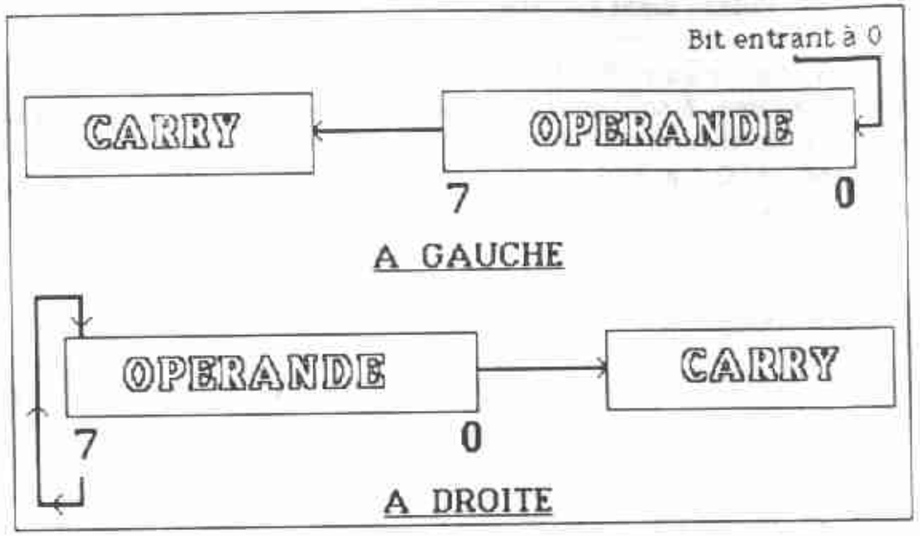


FIGURE 15 : DECALAGES ARITHMETIQUES

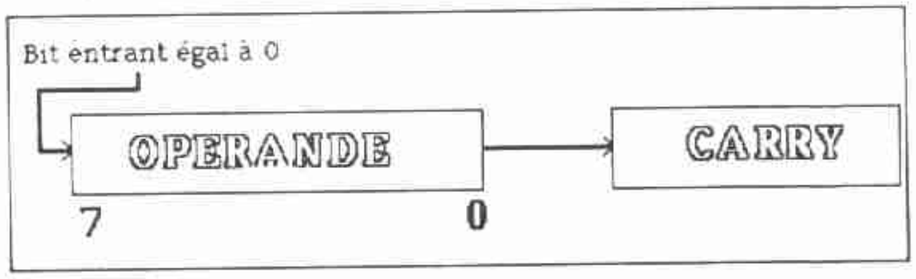


FIGURE 16 : DECALAGES LOGIQUES

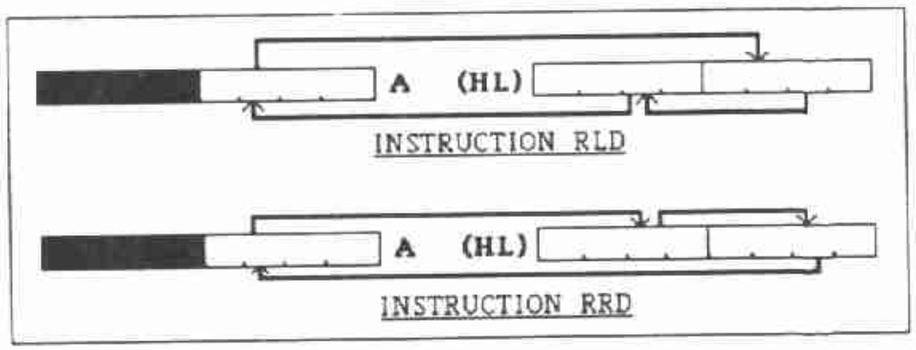


FIGURE 17 : DECALAGES BCD

Avec cette instruction , des décalages ouverts sur **16 bits** peuvent être réalisés .

On obtient les commandes suivantes :

SRL r **SRL (HL)** **SRL (IX + dpl)** **SRL (IY + dpl)**

4.6.5 DECALAGES CIRCULAIRES BCD gauche et droite .

Ce dernier groupe de décalages concerne deux instructions puissantes en arithmétique . En effet , les **décalages BCD** ne se font plus sur un bit comme précédemment mais sur **quatre bits** . L'accumulateur est utilisé comme registre de manoeuvre et l'opérande sera le **pointeur registre HL** (Voir figures 17) .

4.7 INSTRUCTIONS D' USAGE GENERAL et de CONTROLES .

Ce sont des instructions réparties en trois catégories principales :
- Les instructions agissant sur le CARRY .
- Les commandes liées aux interruptions .
- Les opérations "indispensables" .

4.7.1 Les instructions agissant sur le CARRY .

Elles sont au nombre de deux : **SCF** qui force le CARRY à prendre la valeur 1 et **CCF** qui inverse l'état du CARRY .

4.7.2 Les commandes liées aux INTERRUPTIONS .

Une interruption est un événement matériel extérieur provoquant l'interruption du programme en cours d'exécution . Une routine spécifique traite alors l'interruption et le programme interrompu repart . Une interruption peut très bien venir d'un périphérique , une imprimante par exemple qui alerte l'ordinateur de la fin prochaine de son travail .

Les instructions **EI** et **DI** sont utilisées pour activer et désactiver le système de prise en compte des interruptions du Z 80 .

Les commmandes **IM 0** , **IM 1** et **IM 2** effectuent le passage dans les différents modes d'interruption . Ne vous inquiétez pas , le CANON X-07 ne travaille qu'en mode 0 .

4.7.3 Les opérations "INDISPENSABLES" .

Il existe effectivement deux instructions indispensables : **NOP** et **HALT** .

L'instruction **NOP** est très puissante car ... elle ne fait rien !! A quoi peut-elle bien servir alors ? Par exemple , à remplacer les **instructions inutiles** dans le listing source : cela évite de tout réassembler , de changer les adresses , etc ... De plus , la fonction NOP constitue une **excellente temporisation** pour différentes routines ...

L'instruction **HALT** , comme son nom l'indique , sert **à stopper le déroulement du programme** . Ce dernier ne pourra être relancé que par un "RESET" ou une interruption quelconque (Appel de périphérique ...) .

4.8 Les RUPTURES de SEQUENCE .

Ce sont les **"GOTO"** et les **"GOSUB"** du langage BASIC .

4.8.1 Les SAUTS .

Les sauts sont représentés par l'instruction **"JP"** . Cette commande peut être astreinte à une condition particulière définie ci-dessous :

C , Z , M , PE , NC , P , PO , NZ

Le saut sera effectué si le drapeau concerné (Z , C ...) se trouve positionné à 1 . Sinon , le programme continuera à l'instruction suivante . Ainsi , pour l'instruction JP , les possibilités sont les suivantes :

JP nn	**JP NZ , nn**
JP C , nn	**JP NC , nn**
JP Z , nn	**JP PO , nn**
JP M , nn	**JP PE , nn**
JP P , nn	

N'oubliez pas que "nn" représente une adresse codée sur 16 bits (Les sauts peuvent aussi se faire sur 8 bits ...) . De plus , l'adresse de saut peut aussi être fournie par un registre :

JP (HL) **JP (IX)** **JP (IY)**

Ceci permet d'exécuter un saut à une adresse calculée .

De même , les sauts relatifs (instruction **JR**) fonctionne de manière identique à trois restrictions près :

- Les limites des sauts sont fixés à **127** en avant et **128** en arrière .
- **On ne peut utiliser l'adresse se trouvant dans un registre** .
- Les conditions sont beaucoup **plus limitées** .

On obtient tout de même :

JR dpl	**JR NC , dpl**
JR C , dpl	**JR NZ , dpl**
JR Z , dpl	

Enfin , il existe une puissante commande nommée **DJNZ** . C'est une instruction de saut agissant avec **le registre B** . Elle **décrémente le registre B** et **saute à l'adresse indiquée en opérande s'il n'est pas égal à 0** . Par contre , s'il est nul , la prochaine instruction sera exécutée .

4.8.2 Les APPELS de SOUS-PROGRAMMES .

L'instruction **CALL** fait ici office de **"GOSUB"** . Lors d'un CALL , le NSC 800 sauvegarde l'adresse de retour dans la pile . De ce fait , il faut prendre garde à ne pas perdre cette adresse de retour pour que le pointeur puisse retourner au programme principal , une fois la routine terminée . Donc , amateurs en L.M. , ne touchez pas trop à la PILE quand vous appelez un sous-programme !!

Il existe , comme pour l'instruction JP , les CALL suivants :

CALL nn	**CALL NZ , nn**
CALL C , nn	**CALL NC , nn**
CALL Z , nn	**CALL PO , nn**
CALL M , nn	**CALL PE , nn**
CALL P , nn	

4.8.3 Les RETOURS de SOUS-PROGRAMMES .

L'instruction **RET** est assimilable au **"RETURN"** du BASIC . On obtient les possibilités suivantes :

RET	**RET NZ**
RET C	**RET NC**
RET Z	**RET PO**
RET M	**RET PE**
RET P	

4.8.4 Les APPELS de SOUS-PROGRAMMES en PAGE 0 .

Ce sont seulement des **CALL** mais présentant deux différences essentielles par rapport aux CALL précédents :

_ D'une part , le code opératoire de ces commandes n'occupe qu'**un seul octet** contre trois pour les CALL classiques .
_ Ils se réfèrent à des **SOUS-PROGRAMMES d'adresses fixes** , situés en début de mémoire (PAGE 0) .

Les sous-programmes appelés seront implantés aux adresses **0 , 8 , 10h, 18h , 20h , 28h , 30h et 38h** . L'adressage de ces instructions est direct et les indicateurs ne sont pas affectés lors de leur emploi .

Dans le mode 0 du NSC 800 , l'apparition d'une interruption déclenche un appel à l'une de ces adresses contenant des routines d'interruption . Ces commandes portent aussi le doux nom de **ReSTarts** .
En ce qui concerne le X-07 , voici à quoi correspondent tous ces appels systèmes _

-> **RST 0** : Correspond au "BREAK POINT" de la carte moniteur XP 140F .
Un code C9 (RET) est implanté en 0 quand la carte n'est pas en place.

-> **RST 8** : Cette commande compare deux symboles entre eux .
La comparaison se fait entre l'octet pointé par HL et l'octet suivant l'appel par RST 8 : _ Si concordance , retour à RST 8 + 2 et HL=HL+1 .
_ Si désaccord , "SN ERROR" est affiché .

-> **RST 10h** : Cette instruction examine le symbole suivant . Elle charge dans A le caractère pointé par HL .
En sortie , HL incrémenté et : _ C=1 si caractère alphanumérique
_ C=0 si caractère alphabétique

-> **RST 18h** : Cette commande émet un ordre vers le sous-processeur T6834 .
Le registre A doit contenir le code de la commande . Il n'y a pas de paramètres possibles .

-> **RST 20h** : Cette instruction compare les registres DE et HL :
_ Si HL < DE , C=1
_ Si HL > DE , C=0
_ Si HL = DE , Z=1
Seul le registre A est modifié .

-> **RST 28h** : Cette instruction émet le contenu de A vers le dispositif ouvert , en général l'écran LCD . Tous les registres sont préservés .

-> **RST 30h** : Cette instruction teste le type de données se trouvant dans A .
On obtient le tableau récapitulatif suivant :

Adresse 1D9h	TYPE	FLAGS	Contenu de A
2	ENTIER	NC , Z , M	-1
3	CHAINE	Z , C , P	0
4	S. PRE.	NZ , C , P	1
8	D. PRE.	NZ , NC , P	5

-> **RST 38h** : Cette commande calcule l'adresse des tableaux .

4.9 Les ENTREES / SORTIES .

Il n'existe en fait que deux instructions relatives aux entrées / sorties : **IN** et **OUT** . Mais il existe différentes façons de les appeler , ce qui offre beaucoup de possibilités .

Nous avons déjà les commandes **OUT (v),A** et **IN A,(v)** : le contenu du registre A est envoyé à **l'organe périphérique** d'adresse v , dans le cas de l'instruction OUT . Par contre , pour la commande IN , le registre A est chargé avec l'information émise par **le périphérique** d'adresse v .

Nous avons aussi **OUT (C),r** et **IN (C),r** . Ici , l'adresse du périphérique est contenue dans le **registre C** et r peut représenter n'importe quel **registre 8 bits** du NSC 800 .

Enfin , il existe quatre puissantes commandes nommées **INI , IND , OUTI et OUTD** . L'adresse du périphérique se trouve toujours dans le **registre C** mais l'information émise (OUT) ou lue (IN) se trouve **en mémoire (pointée par le registre HL)** et non plus dans un registre comme précédemment .
A chaque exécution de l'une de ces instructions , **le registre B est décrémenté de 1** , le registre HL est incrémenté de 1 pour les instructions INI et OUTI et décrémenté de 1 pour les instructions IND et OUTD .

Les ports de sortie exploités par ces instructions seront explicités dans la deuxième partie de cette ouvrage .

4.10 INSTRUCTIONS sur BITS et CHAINES .

4.10.1 INSTRUCTIONS sur BITS .

Habituellement , le NSC 800 travaille sur des valeurs de 8 bits ou exceptionnellement sur des quartets (groupe de 4 octets) . Les commandes sur bits affinent encore ces opérations avec :

_ Mise à 1 d'un bit : instruction **SET** .
_ Mise à 0 d'un bit : instruction **RES** .
_ Test de la valeur d'un bit : instruction **BIT** .

L'instruction BIT charge dans le drapeau Z l'inverse du bit testé . Les commandes disponibles sont (**b représente le numéro du bit**) :

BIT b , (HL)	**SET b , (HL)**	**RES b , (HL)**
BIT b , (IX + dpl)	**SET b , (IX + dpl)**	**RES b , (IX + dpl)**
BIT b , (IY + dpl)	**SET b , (IY + dpl)**	**RES b , (IY + dpl)**
BIT b , r	**SET b , r**	**RES b , r**

4.10.2 INSTRUCTIONS sur CHAINES .

Ces commandes opèrent sur **des chaines , des zones de mémoire** . On recense trois groupes principaux d'instructions sur chaines : les commandes de **TRANSFERT** , de **COMPARAISON** (ou de **RECHERCHE**) et d'**ENTREES / SORTIES** .

Les instructions de transfert permettent de transférer une zone mémoire d'un endroit à un autre . Si l'on part du début de la zone , on utilise l'instruction **LDIR** . A l'inverse , si l'on part de la fin de la zone , on utilise la commande **LDDR** . Les registres suivants sont utilisés :

HL --------> adresse de la zone source à déplacer .
DE --------> adresse de la zone de destination .
BC --------> longueur de la zone en octets .

On transfert les octets pointés par le registre HL dans la zone pointée par le registre DE et ce , jusqu'à ce que **le registre BC soit nul** . Les registres HL et DE ont été **incrémentés ou décrémentés** durant l'opération , suivant l'instruction LDIR ou LDDR .

Les deux instructions de recherche CPIR et CPDR permettent de rechercher une coincidence entre l'octet contenu dans le registre A et les octets d'une chaine pointée par le registre HL .

Cette opération prend fin lorsque la coincidence attendue a été établie ou lorsque l'on arrive à la fin de la chaine . La zone mémoire peut être scrutée de haut en bas (instruction CPIR) ou de bas en haut (commande CPDR) .

Dans le cas de CPIR , l'accumulateur est comparé avec l'octet pointé par le registre HL . HL est incrémenté et BC se trouve décrémenté . Si la valeur du registre A est égale à la valeur de l'octet pointé , l'opération se termine et le drapeau Z est mis à 1 . Sinon , l'opération se poursuit jusqu'à trouver cette égalité ou la fin de chaine (le registre BC est alors égal à 0 et le drapeau V est mis à 0) .

Les instructions d'entrées / sorties sont au nombre de quatre et permettent de dialoguer avec un périphérique , avec une zone mémoire longue de 1 à 256 octets .

L'adresse du périphérique est contenue dans **le registre C** et l'adresse de la chaine à émettre ou à recevoir se trouve dans **le registre HL** . Comme pour les autres commandes de chaine vues précédemment , l'avant dernière lettre de la mnémonique indique si l'adresse de la chaine doit s'incrémenter ou se décrémenter .

Nous obtenons les commandes suivantes :

OTIR	**OTDR**
INIR	**INDR**

4.11 INSTRUCTIONS de PILE .

Les instructions de **PILE** sont au nombre de deux : **PUSH** et **POP** . La commande PUSH enverra vers la pile le contenu du double registre spécifié . Evidemment , l'instruction POP ira chercher la première valeur se trouvant sur cette pile et la stockera dans le double registre présent en opérande .

Les deux instructions disponibles sont donc :

PUSH rr **POP rr**

rr désigne n'importe quel registre 16 bits sauf le registre PC , bien sûr ...

Ouf !! Vous pouvez souffler !! Nous savons que l'inventaire des instructions d'un micro-processeur est fastidieux mais il est nécessaire . Ne vous affolez surtout pas à la vue de toutes ces commandes ... Bien qu'elles soient très nombreuses , vous apprendrez très rapidement à les maitriser en les utilisant . Cela parait évident mais l'ASSEMBLEUR étant un langage plus ardu à appréhender que le BASIC , il nécessite une longue pratique ainsi qu' une grande rigueur . Passons maintenant aux "PSEUDO INSTRUCTIONS" ...

LES PRINCIPALES PSEUDO-INSTRUCTIONS DE L' ASSEMBLEUR

Ces nouvelles commandes sont un peu particulières . En effet , elles sont **spécifiques** de l'ASSEMBLEUR que vous utilisez . Un ASSEMBLEUR peut donc très bien ne posséder aucune de ces pseudo-instructions .

Ces directives se plient aux mêmes règles syntaxiques que les instructions classiques . Effectivement , leur format source possède les mêmes champs que ceux étudiés au paragraphe 2.5.

Les huit "pseudo-instructions" que nous allons vous exposer s'adressent **uniquement au programme ASSEMBLEUR et non au NSC 800** , comme le font les instructions vues au chapitre 4 . Elles agissent sur le listing source en modifiant sa disposition , son implantation ...

5.1 ADRESSE D' IMPLANTATION : ORG .

La directive **"ORG"** permet , comme son nom l'indique (ORiGine) , d'implanter le programme écrit en ASSEMBLEUR **à partir d'une adresse donnée** . En effet , elle accepte un opérande représenté par une valeur codée sur 8 ou 16 bits .

Ainsi , la pseudo-instruction "ORG 2500h" suivi d'un programme ASSEMBLEUR fixera l'origine de ce dernier à l'adresse 2500h .

Voyons un exemple concret :

```
ORG 4000h
NOP
LD HL,22h
RET
```

Cette petite routine sera implantée à partir de l'adresse 4000h . La case mémoire 4000h contiendra le code de l'instruction NOP (00) .

5.2 RESERVATION DE MEMOIRE : DEFS .

Cette directive a pour fonction de réserver un certain **nombre d'octets** en mémoire . Elle accepte pour opérande des valeurs codées sur 8 ou 16 bits .

Par exemple , la directive "DEFS 350" réservera 350 octets dans le programme suivant :


```
Adresse 2000  :  LD A,5
              :  DEFS 350
        2352  :  LD BC,2250
```

Remarquez qu'après l'adresse 2000 (en décimal) , le code source reprend en 2352 étant donné que DEFS 350 n'est pas assemblé ...

5.3 DEFINITION D' EQUIVALENCE : DEFL et EQU .

Ces deux directives varient souvent suivant l'ASSEMBLEUR utilisé . En effet , **le signe "=" d'affectation est souvent préféré** .

On emploie **"DEFL"** et **"EQU"** quand on désire manipuler des symbo' l'instar des valeurs . Effectivement , la directive "EQU" assigne une immuable à un symbole . Par exemple , "DEBUT EQU 6500h" assigne au symbole "DEBUT" la valeur 6500h . Ceci est équivalent à "DEBUT=6500h" .

Lorsqu'une équivalence est définie , elle reste "intouchable" durant tout le déroulement du programme . Néanmoins , il existe un recours : la directive "DEFL" , employée à la place de "EQU" , permet de contrer cette interdiction .

Le programme suivant vous développe son utilisation :

```
VAL1 EQU 56     ---> Première assignation
LD A,VAL1
LD B,53
VAL1 DEFL 54    ---> Possibilité de changer la première assignation
LD BC,VAL1
VAL1 DEFL 60    ---> Troisième assignation
LD DE,VAL1 .....
```

5.4 DEFINITION de DONNEES : DEFB , DEFM et DEFW .

Ces trois directives se démarquent des précedentes . En effet , elles produisent chacune un code objet contrairement à ORG , DEFL , EQU , DEFL et END .

5.4.1 La directive DEFB .

"DEFB" permet de **définir un octet** (DEFine Byte en anglais) . Par exemple , la ligne "OCTET DEFB 24h" assignera au symbole "OCTET" la valeur 24h .

En effet , si par la suite on découvre la ligne "LD A,(OCTET)" , le registre A ne contiendra pas l'octet pointé par l'adresse 24h mais la valeur 24h elle-même .

5.4.2 La directive DEFM .

"DEFM" a pour fonction **de définir un message** quelconque (DEFine Message en anglais) . Par exemple , la ligne " DEFM 'AU REVOIR' " sera assemblée et produira une donnée 'AU REVOIR' codée en ASCII . Cette donnée pourra ensuite être exploitée par le programme .

Cette directive est très utilisée pour afficher des textes , des tableaux ... sur l'écran ou sur un périphérique quelconque (imprimante , traceur ...) .

5.4.3 La directive DEFW .

"DEFW" permet de définir un mot (DEFine Word en anglais) . Par exemple, la ligne "ADRESSE DEFW 2500h" assignera au symbole "ADRESSE" la valeur 2500h .

En effet , si par la suite l'ASSEMBLEUR découvre la ligne "LD HL,(ADRESSE)" , le double registre HL contiendra l'adresse correspondant au symbole "ADRESSE" , c'est à dire la valeur 2500h .

5.5 FIN du PROGRAMME SOURCE : END .

Cette directive constitue obligatoirement **la dernière ligne source** du logiciel . Sa fonction est évidente : elle indique à l'ASSEMBLEUR **la fin du programme** (comme en BASIC) .

Dans certains cas , "END" possède un opérande constitué par une valeur ou un label . Cela permet de lancer le programme ASSEMBLEUR dès que son chargement est terminé .

```
DEBUT   LD A,5
        LD HL,2020
          .....
        END DEBUT
```

Par exemple , dès que cette routine sera chargée en mémoire et assemblée , elle sera lancée automatiquement par la ligne "END DEBUT" .

A l'image de cette directive ASSEMBLEUR , nous avons terminé ce cinquième chapitre et aussi cette première partie . Quoi ? Qu'entends-je ?? ... Vous désirez des exemples ? Vos désirs étant des ordres , **un sixième et dernier chapitre** va apparaître insidieusement devant vos yeux , vous permettant de bien vous roder à l'ASSEMBLEUR . Après ceci , le CANON X-07 deviendra le maître suprême de cet ouvrage !!

APPLICATION : TRI A BULLES

Dans ce dernier chapitre de la première partie , nous allons "disséquer" **une application écrite en langage machine** qui vous permettra de bien comprendre l'écriture et la forme d'une routine ASSEMBLEUR .

L'application choisie est constituée par une routine de **"tri à bulles"** autorisant **un tri croissant ou décroissant sur des valeurs comprises entre 0 et 255 uniquement** . En effet , un tri sur tous les nombres possibles compliquerait singulièrement les choses et ce n'est pas notre but .

Le listing BASIC , le listing source et le listing désassemblé vont vous être présentés ci-dessous .

LISTING SOURCE

Nous vous rappelons que le listing source constitue le programme construit à l'aide d'un logiciel "ASSEMBLEUR" (Ici , celui de la revue MICRO SYSTEMES : voir annexe 2) . En voici le détail :

```
0 '** TRI DE NOMBRES COMPRIS ENTRE 0 ET 255 **
5 '
10 'ORG &4096
20 '#DE LD B,&0
30 'LD HL,$1025
40 'LD C,(HL)
50 'DEC C
60 '#BB INC HL
70 'INC HL
80 'LD A,(HL)
90 'DEC HL
100 'CP (HL)
110 'JP NC,#AA
120 'LD D,(HL)
130 'LD (HL),A
140 'INC HL
150 'LD (HL),D
160 'LD B,&1
170 'DEC HL
180 '#AA DEC C
190 'JP NZ,#BB
200 'LD A,B
210 'CP &1
220 'RET NZ
230 'JP #DE
240 '
```

Tout d'abord , ce listing **amène quelques remarques** étant donné que chaque ASSEMBLEUR **possède ses particularités** :

_ Cet ASSEMBLEUR utilise **l'éditeur BASIC** pour l'écriture des routines .

_ Chaque routine est précédée du caractère "[" et terminée par le signe "]", ceci pour bien délimiter le champ du programme à assembler .

_ Après chaque numéro de ligne , **le signe d'apostrophe** apparait . Cela peut paraître contradictoire car , en général , c'est le symbole des lignes de remarques . Ici , le symbole des lignes de commentaires est "*" . Le signe d'apostrophe est utilisé par cet ASSEMBLEUR pour différencier les lignes BASIC des lignes à assembler .

_ **Le point décimal** est utilisé à la place de la virgule . De plus , les labels sont symbolisés par **deux lettres précédées du signe "*"** .

_ Enfin , le signe **"$"** assigne la notation hexadécimale à une valeur . De même , le symbole **"&"** assigne la notation décimale à une donnée .

Nous allons pouvoir passer à l'explication du programme :

Ligne 0 : titre de la routine .
Ligne 5 : début de la routine .
Ligne 10 : fixation de l'origine à l'adresse 4096 en décimal (1000h en hexa) .
Ligne 20 : chargement du registre B avec la valeur 0 (label #DE : début) .
Ligne 30 : chargement du registre HL avec l'adresse $1025 (Les octets qui suivent cette adresse $1025 contiennent les données "POKEES" à partir du programme BASIC) .
Ligne 40 : chargement du registre C avec le nombre de données (Cette valeur est "POKEE" à partir du programme BASIC) .
Ligne 50 : on décrémente le registre C d'une unité .
Ligne 60 : on incrémente le registre HL d'une unité (label #BB présent) .
Ligne 70 : on incrémente le registre HL d'une unité .
Ligne 80 : le registre A est chargé avec la donnée contenue dans l'octet pointé par le registre HL .
Ligne 90 : on décrémente le registre HL .
Ligne 100 : on compare la donnée pointée par le registre HL à la donnée contenue dans l'accumulateur .
Ligne 110 : si la donnée pointée par HL est plus petite ou égale à la donnée contenue dans A , on saute au label #AA .
Ligne 120 : sinon , on charge le registre D avec la valeur pointée par HL .
Ligne 130 : on charge l'octet pointé par HL avec le contenu du registre A .
Ligne 140 : on incrémente HL .
Ligne 150 : on charge l'octet pointé par HL avec la donnée contenue dans le registre D .
Ligne 160 : on charge le registre B avec la valeur 1 .
Ligne 170 : on décrémente HL .
Ligne 180 : le label #AA est présent . On décrémente le registre C .
Ligne 190 : on saute au label #BB si le registre A n'est pas égal à la donnée pointée par le registre HL

Ligne 200 : sinon , le registre A est chargé avec la valeur contenue dans B .
Ligne 210 : le registre A est comparé à la valeur 1 .
Ligne 220 : si le tri est terminé (B=0) , alors on retourne au BASIC .
Ligne 230 : sinon , on repart au label #DE pour continuer le tri .
Ligne 240 : fin de la routine ASSEMBLEUR .

LISTING DESASSEMBLE

Le listing désassemblé est constitué par les codes rentrés en mémoire transformés en mnémoniques . Ce travail est opéré par un DESASSEMBLEUR (Voir annexe 2) .

Adresse (HEXA)	Adresse (DECIMAL)	Mnémonique
1000	4096	LD B,00
1002	4098	LD HL,1025
1005	4101	LD C,(HL)
1006	4102	DEC C
1007	4103	INC HL
1008	4104	INC HL
1009	4105	LD A,(HL)
100A	4106	DEC HL
100B	4107	CP (HL)
100C	4108	JP NC,1016
100F	4111	LD D,(HL)
1010	4112	LD (HL),A
1011	4113	INC HL
1012	4114	LD (HL),D
1013	4115	LD B,01
1015	4117	DEC HL
1016	4118	DEC C
1017	4119	JP NZ,1007
101A	4122	LD A,B
101B	4123	CP 01
101D	4125	RET NZ
101E	4126	JP 1000

LISTING du PROGRAMME BASIC

Nous allons utiliser un programme écrit en BASIC pour charger les codes de la routine écrite en langage machine . De plus , ce programme nous permettra de rentrer le nombre de données ainsi que les données une à une . Le tri peut être croissant ou décroissant et les nombres seront affichés dès que vous aurez choisi votre classement .

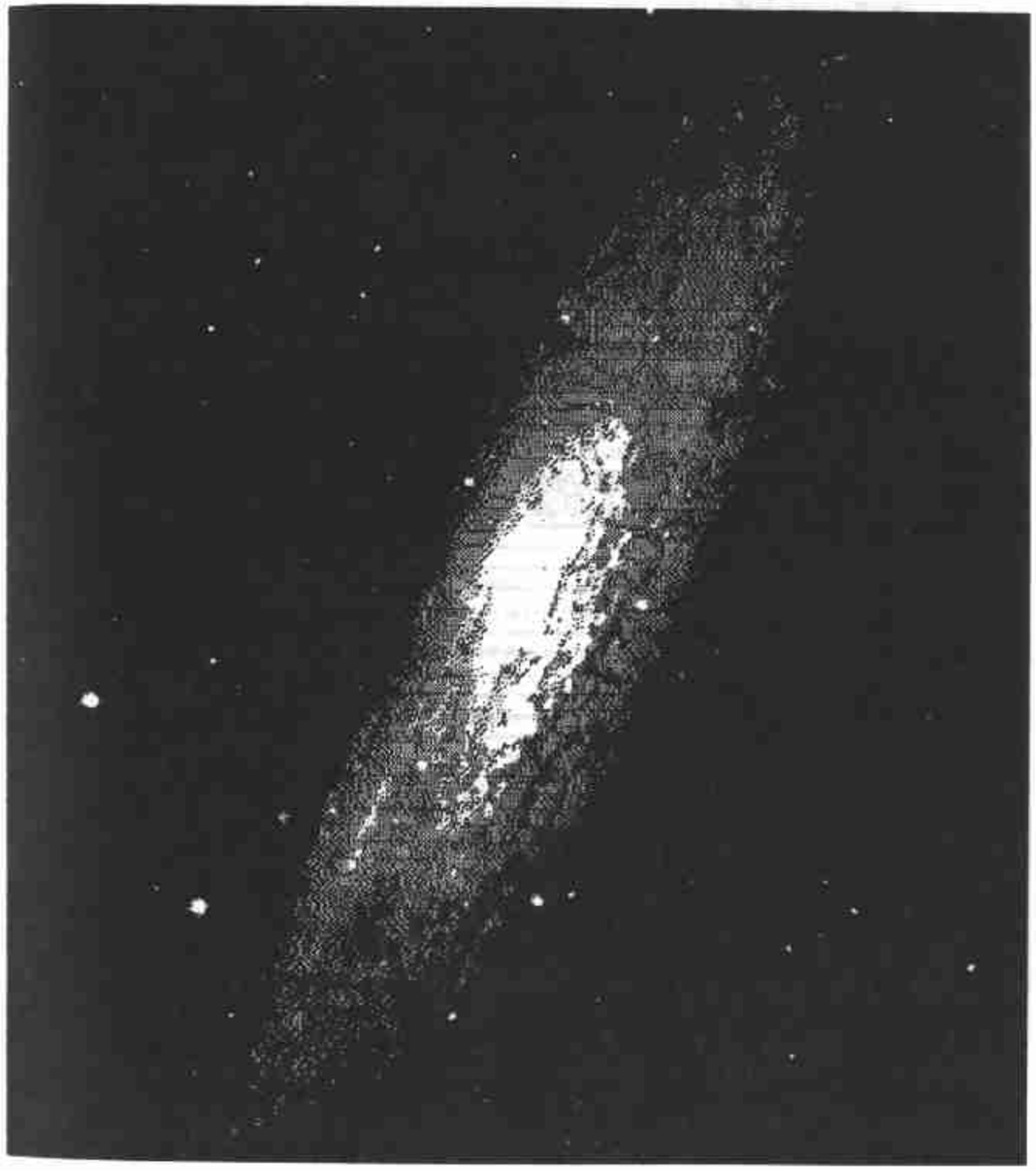
```
10 ' TRI CROISSANT ET DECROISSANT
20 ' NOMBRES POSITIFS INFERIEURS A 255
30 DATA 6,0,21,25,10,4E,D,23,23,7E,2B,BE,D2,16,10,56,77,23,72,6,1,2B,D,C2,7
40 DATA 10,78,FE,1,C0,C3,0,10 : RESTORE 30 : FOR I= 4096 TO 4128 : READ A$
50 POKE I,VAL ("&H"+A$) : NEXT I
55 CLS : CLEAR 50,4050 : INPUT "Nombre de données ";N : POKE &H1025,N
60 FOR I=&H1026 TO &H1025+N : BEEP 9,2 : PRINT "Donnée";I-&H1025; :
   INPUT A
70 POKE I,A : NEXT I
80 CLS : EXEC &H1000 : PRINT "(C)roissant ou .....(D)écroissant.... ?"
90 G$=INKEY$ : IF G$="D" THEN 120
100 IF G$<> "C" THEN 90
110 CLS : FOR I = &H1026 TO &H1025+N : PRINT PEEK(I); : NEXT I :BEEP 9,2 :
    END
120 CLS : FOR I=&H1025+N TO &H1026 STEP -1 : PRINT PEEK(I); : NEXT I :
    BEEP 9,2 : END
```

Ce petit programme va vous être explicité ci-dessous :

Ligne 10 : commentaire .
Ligne 20 : commentaire .
Ligne 30 : données concernant la routine en L.M.
Ligne 40 : fin des données . Boucle permettant de charger les codes en mémoire , de l'adresse 4096 à 4128 .
Ligne 50 : fin de la boucle .
Ligne 55 : le nombre de données est demandé . Cette valeur est "POKEE" à l'adresse &H1025 . Le CLEAR permet de réserver de la place pour la routine écrite en ASSEMBLEUR .
Ligne 60 : entrée des différentes données .
Ligne 70 : stockage de ces données en mémoire .
Ligne 80 : le tri est lancé au moyen de la commande EXEC .
Ligne 90 : pour un tri décroissant , on va en ligne 120 .
Ligne 100 : controle des entrées .
Ligne 110 : affichage du tri croissant .
Ligne 120 : affichage du tri décroissant .

Voilà !! Cette application type vous permet de juger de **la rapidité du langage machine par rapport au BASIC** . Vous serez sans doute surpris de noter la surprenante vélocité de ce tri : à peine avez-vous tapé sur la lettre "D" ou "C" que les nombres triés s'affichent à l'écran !!

De nombreuses autres applications vous attendent tout au long de **la troisième partie** de ce livre . Mais , avant de vous lancer à corps perdu dans les nombreux programmes que nous vous avons concocté , la deuxième partie de cet ouvrage va vous entrainer **au coeur même de votre CANON X-07** . En effet , vous allez découvrir ses recoins les plus cachés ...



VOYAGE AU COEUR DU CANON X-07

2ème PARTIE :

MYSTERES DU X-07

L' ARCHITECTURE INTERNE DU CANON X-07

7.1 GENERALITES .

Le CANON X-07 a été construit avec des circuits à **technologie CMOS** (faible consommateurs d'energie) , étant donné sa finalité de portable . Il est , de ce fait , alimenté par quatre piles de 1,5 Volts .

Si cette technologie CMOS est très économe en électricité , elle est par contre **très lente** dans son travail !! Par conséquent , les ingénieurs de la société CANON ont adopté une architecture un peu particulière pour le X-07 ...

En effet , si le microprocesseur du CANON X-07 est bien un Z-80 (ou plutôt un NSC 800 , simili Z-80) , **il existe un deuxième microprocesseur chargé de la gestion des entrées/sorties (clavier , buzzer , affichage ...)** . Ce **"T6834"** , microprocesseur de TOSHIBA , permet de soulager le X-07 de fonctions importantes et la vélocité du CANON s'en trouve améliorée d'autant .

Dans notre machine préférée , le NSC 800 envoie simplement des commandes au T6834 à travers un circuit spécialisé dans les transmissions nommé **"HD61L202F"** .

Une fois que le T6834 a signalé qu'il a bien reçu les commandes envoyées, le microprocesseur principal peut continuer l'exécution du programme en cours pendant que les entrées/sorties s'effectuent .

Le **bloc-diagramme** , présent à la page 66 (figure 18) , nous permet de survoler cette organisation . En fait , tout se passe comme si un ordinateur construit autour d'un Z-80 transmettait des ordres à un ordinateur équipé d'un T6834 .

Afin de vous préparer à l'exploration systématique des ressources cachées du X-07 , nous allons vous exposer l'architecture des deux parties du X-07 : celle attachée au NSC 800 et celle obéissant au T6834 .

7.2 L' UNITE CENTRALE .

L'unité centrale contient un bloc de mémoire directement accessible à partir du BASIC du X-07 par les commandes PEEK et POKE . En effet , la mémoire LCD (Par exemple ...) ne nous est pas accessible car elle est directement adressée par le T6834 .

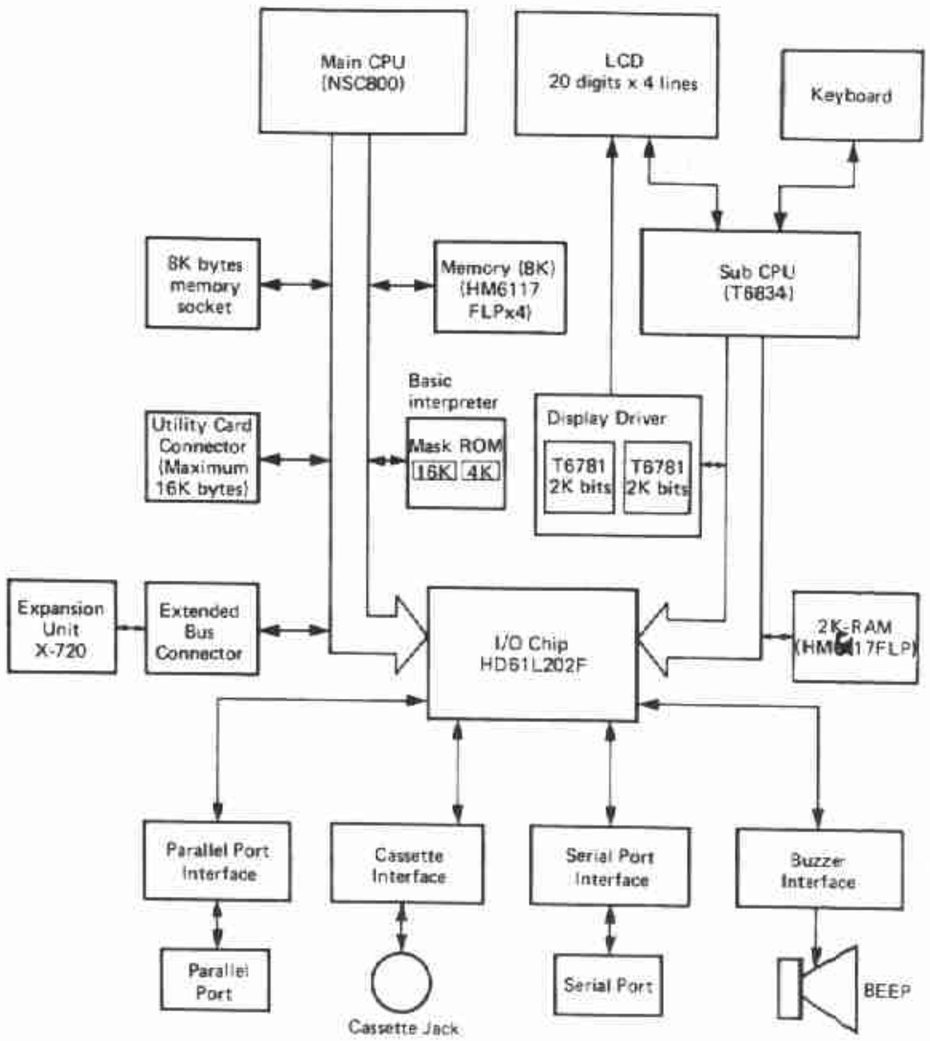


FIGURE 18 : BLOC-DIAGRAMME

Par conséquent , pour que les programmeurs intrépides de la caste CANON puissent utiliser cette étonnante mémoire LCD (Nous en verrons l'utilité dans la troisième partie ...) , il va falloir qu'ils demandent au T6834 d'envoyer certains codes de controle afin d'y écrire . Eh oui , le Z-80 ne sait pas écrire dans cette mémoire !!

L'unité centrale comprend :

_ Un microprocesseur Z-80 .
_ 8 Kilo-octets de RAM en version de base .
_ 20 Kilo-octets de ROM contenant l'interpréteur BASIC .

On peut , en outre , y adjoindre :

_ 16 Kilo-octets de RAM (sous forme de CHIP ou de carte mémoire) .
_ 8 Kilo-octets de ROM (carte préprogrammée CANON) .
_ 4 Kilo-octets de ROM (interface vidéo X-720) .
_ 16 Kilo-octets de RAM Vidéo .

7.3 Le PLAN de la MEMOIRE .

Le NSC 800 peut adresser **64 Kilo-octets de mémoire** (comme tous les processeurs 8 bits ...) répartis en plusieurs secteurs décrits à la figure 19 .

Deux choses particulièrement importantes sont à noter :

_ L'adressage de la carte mémoire et de la prise est **inversé** lorsque l'on place une mémoire additionnelle de 8 Kilo-octets . Ceci permet d'obtenir un adressage continu lorsqu'une carte est retirée .

_ On note que la partie basse de la mémoire est divisée en **deux zones** . Pourquoi ? A la mise sous tension du X-07 , le NSC 800 , comme tous les processeurs , cherche à effectuer les instructions se situant à **l'adresse 0000** . Il est donc absolument vital qu'un programme s'y trouve ... A la mise en route du CANON , cette zone mémoire est constituée par de la RAM puis le X-07 la "masque" et le bas de la mémoire redevient de la RAM .

7.4 La ROM des CARTES MEMOIRE .

Lorsque l'on place une carte mémoire dans le logement destiné à cet effet, il est primordial que le X-07 la détecte pour que l'on puisse s'en servir .

Une fois la carte détectée , le CANON modifie la **zone "RAM système"** (Pour en savoir plus sur cette zone , reportez-vous au chapitre 8) .

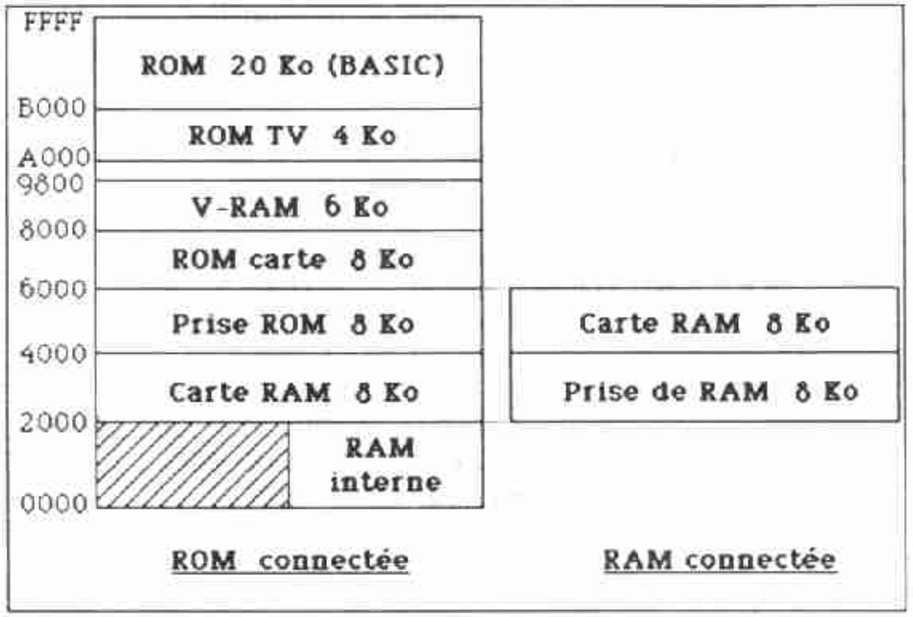


FIGURE 19 : STRUCTURE - MEMOIRE

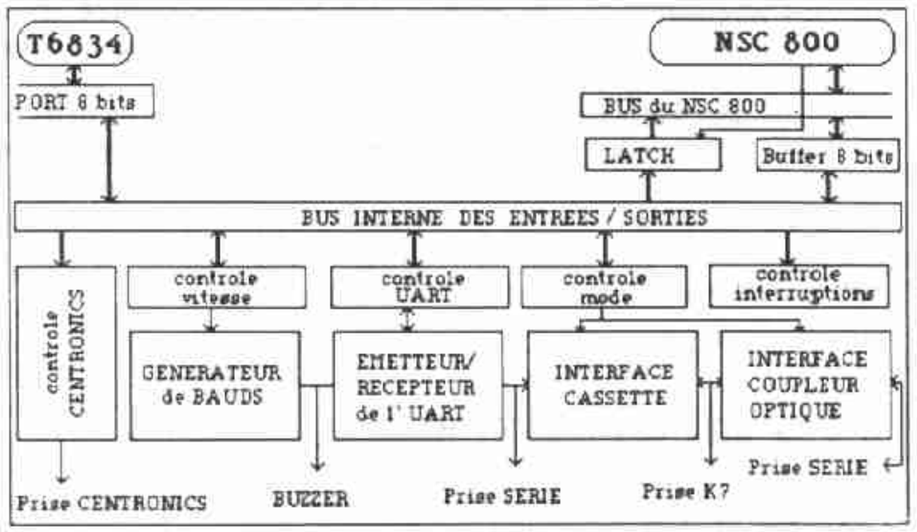


FIGURE 20 : les ENTREES / SORTIES

La carte étant détectée par voie logicielle , le programme d'initialisation recherche **un mot-clé baptisé "love"** dans la mémoire . Dès que ce mot-clé est repéré , on considère **les six octets** le suivant comme les adresses de trois sous-programmes à exécuter immédiatement avant de continuer à effectuer le programme d'initialisation .

En fait , le X-07 exécute toujours les routines correspondant aux deux premières adresses , la troisième servant lorsque le CANON a été éteint par l'ordre SLEEP du BASIC .

Pour éviter toute confusion (En effet , rien ne prouve que vous n'écrivez jamais le mot "love" dans la mémoire du CANON ... Sait-on jamais , l'amour peut prendre toutes les formes !!) , le X-07 ne recherche pas ce mot-clé dans toute la mémoire , mais seulement entre **les adresses 2000h et B000h et ceci tous les 2048 octets (800 en héxadécimal)** .

Nous vous conseillons fortement de vous reporter au **programme de "COPYRIGHT personnel" de la troisième partie** qui vous permettra de bien comprendre ce processus . En effet , comme vous le verrez , rien ne vous empêche de placer le mot "love" avant celui de la version de base afin de faire exécuter par le CANON vos propres routines (L'imagination est au pouvoir !!) .

7.5 La PUCE de CONTROLE "ENTREES/SORTIES" .

Cette puce constitue **un circuit spécialisé controlant la réception et l'exécution des données** .

Bien qu'étant relativement inintéressante pour la programmation du CANON X-07 , son architecture et son mode de travail va vous permettre de mieux appréhender le fonctionnement de votre micro-ordinateur .

Son bloc-diagramme est représenté par la figure 20 . En regardant de près ce schéma , on note la grande puissance et la tache ardue de ce circuit qui doit gérer **les sorties CENTRONICS , les entrées/sorties SERIE , le BUZZER et l'interface cassette** .

Le NSC 800 communique avec ce circuit via **les ports d'adresse F0 à F7** . Comment ? Tout simplement en écrivant dans les registres de notre transistor HD61L202F , décrit précédemment .

Voici la liste des informations lues et écrites avec l'adresse du port correspondant . Le mode d'adressage de ces registres est celui utilisé normalement par le NSC 800 . Le port est spécifié par l'adresse de l'opérande , D7 à D0, correspondant au bus de données .

(Ports)	D7	D6	D5	D4	D3	D2	D1	D0	(Bits)
F0	CONTROLE DES INTERRUPTIONS								
	OFFRQ	SETPS	BIE	ALMIE	XTIE	XRIE	TIE	RIE	
F1	XBTR : STOCKAGE DES INFORMATIONS POUR C.C.U								
F2	CONTROLE DE BAUDS - POIDS FAIBLE								
F3	CONTROLE DE BAUDS - POIDS FORT								
F4	MODES								
	SETBC	BRGST	CNTR	LEO	MD1	MD0	BZON	REM	
F5	INTERRUPTION (RESET)								
	RALRM	CSTB	CSFT	RBGI	RTDRE	RRDRF	RXTRE	RXRRF	
F6	CONFIGURATION de l' UART								
		IR	RTS	ER	SBRK	RXE		TXEN	
F6'	TRANSMISSION SERIE								
	S2		EP	PEN		L1	B2	FX04	
F7	CONTROLE de l' UART								

FIGURE 21 : DONNEES ECRITES

MD1	MD0	Mode
0	0	RESET - Le compteur de bauds s'arrête
0	1	Mode SERIE
1	0	Mode K7 - Le signal sort sur la prise
1	1	Le son sort sur le BUZZER

FIGURE 22 : MD1 et MD0

7.5.1 Le mode ECRITURE .

Le schéma représenté par la figure 21 exposent les registres du HD61L202F dans lesquels le NSC 800 écrit les informations .

Voici la signification des symboles utilisés dans cette figure et le détail de l'utilisation de chaque port .

Le port F0 controle les interruptions . Il faut noter que la modification des données inscrites dans ce port risque de bloquer le CANON X-07 . Voici la signification des différents symboles :

- _ OFFRQ : demande la coupure d'alimentation .
- _ SETPS : entrée en mode d'alimentation économique (X-07 éteint) .
- _ BIE : le générateur de vitesse est autorisé à émettre des interruptions .
- _ ALMIE : autorisation des interruptions provenant de l'alarme .
- _ XTIE : autorisation des interruptions provenant du bus d'extension .
- _ TIE : autorisation des interruptions provenant de la transmission des données (SERIE) .
- _ RIE : autorisation des interruptions provenant de la réception de données (SERIE) .

Le port F1 stocke les informations transmises au sous-processeur .

Les ports F2 et F3 indiquent les vitesses de transmission . Le compteur de bauds étant codé sur 12 bits , on obtient les 8 bits de poids faible dans le port F2 et les 4 bits de poids fort dans les 4 bits de poids faible du port F3 (Soulignons que les 4 bits de poids fort du port F3 ne servent pas dans ce cas précis ...) .

Le port F4 s'occupe du mode des transmissions . Les symboles sont les suivants :

- _ SETBC : au niveau haut (bit à 1) , il permet la mise en place dans le générateur de bauds des informations contenues dans les ports F2 et F3 .
- _ BRGST : au niveau haut , le générateur de bauds compte ... Dès qu'il atteint 0 , ce générateur est stoppé .
- _ CNTR : stockage d'un bit allant être envoyé à la prise parallèle .
- _ LEO : au niveau haut , le signal de 38,4 KHZ est transmis au coupleur optique .
- _ MD1 et MD0 : utilisés afin de forcer le mode de transmission comme à la figure 22 .
- _ BZON : un signal est émis par le BUZZER .
- _ REM : quand ce bit est à 1 , le chiffre 1 est placé sur le canal REM .

Le port F5 sert à réinitialiser les interruptions .

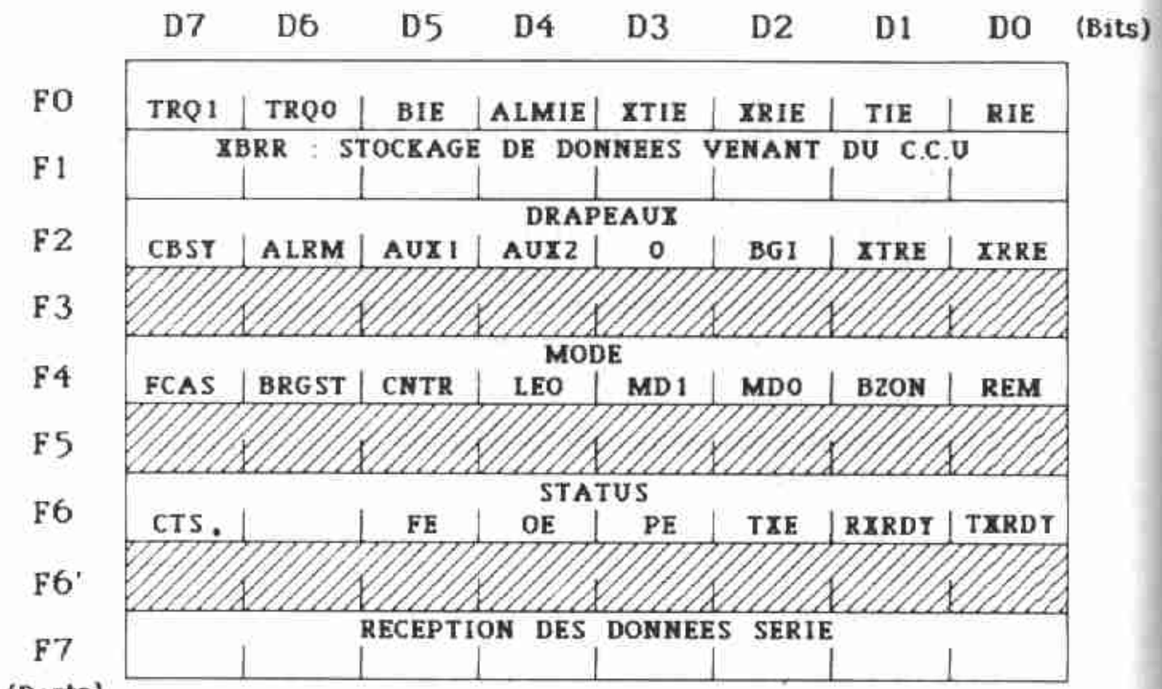


FIGURE 23 : DONNEES LUES

Le port F6 controle l'UART donc la transmission et la réception série . Voici la description des symboles :

- _ IR : initialisation de l'UART .
- _ RTS : transmission du signal de controle quand ce bit se trouve à 1 . La patte RTS du HD61L202F est mise à 0 .
- _ ER : initialisation des drapeaux d'erreurs (PE , OE et FE) .
- _ SBRK : la transmisson du chiffre 0 est forcé même si des données sont présentes quand ce bit est à 1 .
- _ R*E : autorise la réception s'il est mis à 1 . S'il est mis à 0 , la réception n'est pas autorisée .
- _ T*EN : autorise la transmission s'il est mis à 1 . S'il est mis à 0 , la transmission n'est pas autorisée .

Le port F7 stocke les données allant être transmises .

7.5.2 Le mode LECTURE .

Les ports où se trouvent les informations sont lus par le NSC 800 . Reportez-vous à la figure 23 pour les détails . Voici les explications port par port ...

Le port F0 controle toujours les interruptions . Seulement deux bits ont une signification différente :

- _ TRQ0 : ce bit est mis à 1 si la trouche BREAK est enfoncée .
- _ TRQ1 : si ce bit est mis à 1 par le T6834 , le NSC 800 ne peut entrer en mode "sauvegarde" .

Le port F1 stocke les informations transmises par le T6834 .

Le port F2 contient des drapeaux informant le NSC 800 des différents états du CANON X-07 . Voici la signification de ces divers symboles :

- _ CBSY : ce bit est mis à 1 par le signal indiquant que le port parallèle est occupé .
- _ ALRM : ce bit controle l'état des piles . Il est d'ailleurs passé au crible toutes les quatre minutes par le sous-processeur . S'il se trouve à 1 , les piles sont épuisées .
- _ AUX2 : controle de la pile d'une carte mémoire insérée dans le X-07 .
- _ AUX1 : ce bit est mis à 0 quand l'interrupteur de la prise d'extension est placé sur la position "RAM" .
- _ BGI : ce bit est accordé au signal de sortie du générateur de bauds .
- _ XTRE : ce bit est mis à 1 quand le T6834 lit le registre XBTR .
- _ XRRF : ce bit est mis à 1 quand le T6834 écrit dans le registre XBRR .

Le port F4 fonctionne exactement comme en mode écriture sauf pour deux signaux décrits ci-après :

_ FCAS : ce bit représente le reflet de ce que lit le CANON sur une bande magnétique .
_ REM : télécommande du magnétophone .

Le port F6 s'occupe de la gestion des erreurs lors des transmissions sérielles . Voici l'explication des différents symboles :
_ CTS : ce bit vérifie le signal CTS du port SERIE . Si le bit CTS est égal à 0 et le bit T*EN égal à 1 , les données sérielles sont transmises .
_ FE : ce bit est mis à 1 s'il manque un bit de STOP .
_ OE : ce bit est mis à 1 si des informations sont recues avant que le X-07 ait pu lire les précédentes .
_ PE : ce bit est mis à 1 en cas d'erreur de parité .
_ T*E : ce bit est mis à 1 quand le registre de transmission ne contient pas les informations attendues .
_ T*RDY : réception prête .
_ R*RDY : transmission prête .

Le port F7 stocke les données sérielles reçues .

Notons que **les ports F3 et F5** ne servent à rien présentement .

Avec toutes ces informations , il est possible de réaliser à peu près n'importe quel logiciel de transmission de données .

Il est vrai que les pages décrivant les ports sont un peu techniques donc ardues pour les néophytes . Nous vous conseillons donc de ne pas trop vous attarder sur les ports si vous êtes débutant en ASSEMBLEUR . En effet , il sont plutôt utilisés en programmation avancée et permettent des foules de choses .
La puissance se paie en difficulté de compréhension mais une fois que vous maitriserez mieux les mystères du CANON X-07 , vous pourrez revenir fouiller dans ces ports ...

Allons découvrir ensemble le sous-processeur T6834 ... en détail !!

7.6 Le SOUS-PROCESSEUR "TOSHIBA 6834" .

Comme nous l'avons vu précédemment , ce deuxième processeur gère le clavier , l'affichage ainsi que les caractères graphiques , l'horloge , l'alarme , l'alimentation et le controle des piles .

Il dispose de **70 commandes transmises par le NSC 800 via le circuit HD61L202F** . Ces instructions sont souvent suivies de paramètres et , si nécessaire , le T6834 répond par une suite d'octets .

7.6.1 Les COMMANDES ACCESSIBLES .

Les commandes accessibles sont très nombreuses (**70 exactement**) et permettent de multiples réalisations . Elles vous sont données ci-dessous avec quelques explications . Notons que chaque commande possède un code de controle à rappeler dès son utilisation dans un logiciel usant de la programmation avec le T6834 .

CODE (HEXA)	COMMENTAIRES
01	L'instruction TIME CALL renvoie 8 octets correspondant à l'année (2 octets) , le mois , la date , le jour de la semaine , l'heure , les minutes et les secondes .
02	L'instruction STICK renvoie un octet correspondant à la touche de curseur enfoncée . Aucun curseur enfoncé : 30 est renvoyé . Curseur haut enfoncé : 31 est renvoyé . Curseur droit enfoncé : 32 est renvoyé . Curseur gauche enfoncé : 37 est renvoyé . Curseur bas enfoncé : 36 est renvoyé .
03	L'instruction STRIG renvoie 0 si la touche F6 est enfoncée sinon elle renvoie FFh .
04	L'instruction STRIG1 renvoie 0 si la barre d'espace est enfoncée sinon elle renvoie FFh .
05	L'instruction RAM READ permet au NSC 800 de lire la mémoire du T6834 . Lorsque les octets haut et bas de l' adresse sont spécifiés après la commande , le contenu de l'adresse est envoyé à l'accumulateur .
06	L'instruction RAM WRITE permet au NSC 800 d'écrire des données dans la mémoire réservée au T6834 . Lorsque les octets haut et bas de l'adresse sont spécifiés après la commande , les données sont écrites à l'adresse spécifiée de la mémoire du T6834 .
07	L'instruction SCROLL SET , suivie de deux octets , fixe la première et la dernière ligne de roulement de l'écran LCD .
08	L'instruction SCROLL EXET utilisée sans paramètre fait dérouler l'écran . On note que le curseur ne bouge pas .
09	L'instruction LINE CLEAR efface la ligne spécifiée par l' octet suivant la commande .

0A L'instruction TIME SET règle le temps . Les éléments du temps (seconde , minute , heure , jour , date , mois et année) sont envoyés séquentiellement après l'instruction . A noter que l'année est codée sur deux octets .

0B L'instruction Calcul et Réglage du jour de la semaine pose le jour de la semaine correspondant à la date réglée via la commande TIME SET .

0C L'instruction Réglage des données d'Alarme pose les données de l'alarme . Cette commande est programmée de la même façon que la commande 0A . On note que le deuxième paramètre spécifié n'a aucune signification .

0D L'instruction BUZZER OFF fait cesser le BUZZER lorsque l' heure d'alarme est atteinte .

0E L'instruction BUZZER ON fait retentir le BUZZER lorsque l' heure d'alarme est atteinte .

0F L'instruction Transfert de Bit IMAGE sort la ligne spécifiée de l'écran LCD sous le format "bit image" (120 octets) . Le chiffre (compris entre 0 et 3) suivant la commande constitue le numéro de ligne .

10 L'instruction POINT teste la présence d'un point allumé sur l'écran LCD . Le chiffre 0 est renvoyé si un point se trouve aux coordonnées (X,Y) sinon FFh est renvoyé .

11 L'instruction PSET allume le point de coordonnées (X,Y) .

12 L'instruction PRESET éteint le point de coordonnées (X,Y) .

13 L'instruction PEOR inverse le point de coordonnées (X,Y) .

14 L'instruction LINE allume tous les points d'une ligne comprise entre les coordonnées (X_1,Y_1) et (X_2,Y_2) . Donc , 4 octets doivent maintenant suivre la commande dans un ordre X_1 , Y_1 , X_2 , Y_2 .

15 L'instruction CIRCLE trace un cercle de rayon r autour du centre de coordonnées (X_1,Y_1) . Les données devant suivre la commande sont X_1 , Y_1 et r .

16 L'instruction USER DEFINE KEY WRITE assigne des définitions aux touches de fonction F1 à F12 .
On spécifie après la commande le nombre UDK (1 à 12) , la chaine constituant la définition (les 3 caractères de repérage et 38 caractères effectifs) et la chaine "00" indiquant la fin .
On peut noter que 42 caractères effectifs peuvent être programmés avec les touches F6 et F12 .

17 L'instruction USER DEFINE KEY READ est utilisée par le NSC 800 pour lire les touches définies par l'utilisateur . Lorsque le nombre UDK est spécifié , 255 caractères sont renvoyés à partir de l'adresse de début du codage .

18 L'instruction UDK ON provoque le début d'envoi de UDK par le T6834 .

19 L'instruction UDK OFF provoque l'arrêt d'envoi de UDK par le T6834 .

1A L'instruction USER DEFINE CHARACTER WRITE permet de définir des caractères affichables pour les codes de caractères 80 à 9F et E0 à FF .
Il faut introduire le code du caractère et 8 octets de données de caractères (6 points horizontalement * 8 points verticalement) après la commande .

1B L'instruction USER DEFINE CHARACTER READ est utilisée pour lire les définitions de la commande 1A . Les codes de caractères doivent se situer entre 20h et FFh .

1C L'instruction UDC INT renvoie à UDC .

1D L'instruction START PROGRAM WRITE définit un logiciel de lancement pouvant atteindre 511 octets . Le début du programme est posé à l'adresse de départ lorsque les données de chaine et une terminaison ("00") sont introduites après la commande .

1E L'instruction SP WRITE CONT écrit après la commande 1D .

1F L'instruction SP ON permet l'exécution du programme de lancement .

20 L'instruction SP OFF invalide le programme de lancement .

21 L'instruction SP READ lit le programme de lancement .

22 L'instruction ON STATE permet d'obtenir le OFF , ON et le SLEEP ON (Bit 0=1 --> OFF,ON / Bit 6=1 --> SLEEP ON) .

23 L'instruction OFF REQUEST demande la coupure pure et simple de l'alimentation .

24 L'instruction LOCATE déplace la curseur sur la position spécifiée et affiche le caractère . On note que si "00" est spécifié , le curseur est déplacé mais aucun caractère n'est affiché . On doit faire suivre la commande des coordonnées (X,Y) classiques et du code du caractère à afficher .

25 L'instruction Affichage du Curseur allume le curseur .

26 L'instruction Non affichage du Curseur éteint le curseur .

27 L'instruction TEST KEY permet de tester une touche .
Lorsque la ligne d'échantillonage de touches (K) est spécifiée (2 octets) après la commande , la valeur d'échantillonnage de touches OR est renvoyée .
Pour plus de détails , voir la matrice de clavier (figure 25).

28 L'instruction TEST CHR renvoie 0 si la touche du caractère envoyé est enfoncée sinon FFh est répondu . Le caractère spécifié (code ASCII) doit pouvoir être spécifié dans le mode alphanumérique sans utiliser la touche "SHIFT" .

29 L'instruction Remise à Zéro des Secondes initialise les secondes .

2A L'instruction Réglage des données de la Date écrit les données de la date .

2B L'instruction Effacement de l'affichage coupe le signal commun de l'écran LCD . La MEV d'affichage peut être lue ou écrite mais son contenu ne sera pas affiché .

2C L'instruction Allumage de l'affichage rétablit l'affichage de l'écran LCD après la commande 2B .

2D L'instruction KEY BUFFER CLEAR vide le registre "tampon de touches" , sans affecter UDK et SP .

2E L'instruction CLS permet de "nettoyer" l'écran LCD .

2F L'instruction HOME déplace le curseur à (0,0) .

30 L'instruction Affichage de UDK affiche les labels de touches , définies par l'utilisateur , à la ligne 3 .

31 L'instruction Non affichage de UDK éteint l'affichage des précédents labels .

32 L'instruction REPEAT KEY ON valide l'entrée répétée .

33 L'instruction REPEAT KEY OFF invalide l'entrée répétée .

34 L'instruction Réglages des caractères KANA assigne les caractères KANA à UDK .

35 L'instruction UDK CONT WRITE ajoute des données . . chaine de caractères actuellement assignée à l'une des touches de fonction .

36 L'instruction ALARM READ lit les réglages d'alarme . Son format est le même que la commande 01 .

37 L'instruction BUZZER ZERO renvoie 0 si le nombre est 0 sinon renvoie FFh .

38 L'instruction CLICK OFF déclenche la fonction "click" .

39 L'instruction CLICK ON enclenche la fonction "click" .

3A L'instruction LOCATE CLOSE ramène le curseur dans les limites de défilement lorsque ce dernier a été déplacé hors des limites grâce à la commande 24 .

3B L'instruction Validation de Sortie de touche d'unité centrale valide l'entrée de touches .

3C L'instruction Invalidation de Sortie de touche d'unité centrale invalide l'entrée de touches .

3D L'instruction Programme de Lancement provoque l'exécution du programme de lancement dès l'enclenchement de l'alimentation .

3E L'instruction Initialisation du Programme de Lancement provoque l'inexécution du programme de Lancement à l'enclenchement de l'alimentation .

3F — L'instruction REQUEST SLEEP coupe l'alimentation du X-07 et le CANON se trouve en mode SLEEP .

40 — L'instruction UDK INIT rétablit les assignations par défaut des touches de fonction .

41 — L'instruction Ecriture de caractère affiche le motif de 6*8 points spécifié par l'utilisateur sur la position du curseur . Le curseur avance ensuite sur la position suivante . Le motif est composé de 8 octets de données suivant la commande . Le format reste le même que celui de la commande 1A .

42 — L'instruction Lecture de caractère fait sortir 8 octets de données sur la position du curseur .

43 — L'instruction SCANR renvoie 8 bits de données indiquant les réglages des points dans la zone 6*8 (emplacement du curseur) . Le format reste le même que celui de la commande 1B .

44 — L'instruction SCANL renvoie le nombre de points posés à gauche de la position (X-1,Y) .

45 — L'instruction TIME CHK controle les réglages des données de temps vus précédemment suivant les lois normales .
0<=seconde<60 sinon Bit 0=1 (Etat d'erreur ...)
0<=minute<60 sinon Bit 1=1
0<=heure<24 sinon Bit 2=1
0<=jour<=28,29,30,31 sinon Bit 3=1
0<=mois<=12 sinon Bit 4=1
1901<=année<=2099 sinon Bit 5=1
On note que Bit 6=Bit 7=0

46 — L'instruction ALM CHK controle les réglages d'alarme .

La figure 24 récapitule chaque numéro de fonction accompagné de son nombre d'octets (commande et opérateur) et de sa longueur de réponse .

Nous étudierons ensuite **le secteur mémoire du T6864** , zone réservée exclusivement au sous-processeur ...

N°	Nombre d'octets	Réponse (octets)	N°	Nombre d'octets	Réponse
01	1	8	26	1	0
02	1	1	27	3	1
03	1	1	28	2	1
04	1	1	29	1	0
05	3	1	2A	2	0
06	4	0	2B	1	0
07	3	0	2C	1	0
08	1	0	2D	1	0
09	2	0	2E	1	0
0A	9	0	2F	1	0
0B	1	0	30	1	0
0C	9	0	31	1	0
0D	1	0	32	1	0
0E	1	0	33	1	0
0F	2	120	34	1	0
10	3	1	35	2+n+"00"	0
11	3	0	36	1	8
12	3	0	37	1	1
13	3	0	38	1	0
14	5	0	39	1	0
15	4	0	3A	1	0
16	2+n+"00"	0	3B	1	0
17	2	n+"00"	3C	1	0
18	1	0	3D	1	0
19	1	0	3E	1	0
1A	10	0	3F	1	0
1B	2	8	40	1	0
1C	1	0	41	9	0
1D	1+n+"00"	0	42	1	8
1E	1+n+"00"	0	43	3	2
1F	1	0	44	3	2
20	1	0	45	1	1
21	1	n+"00"	46	1	1
22	1	1			
23	1	0			
24	4	0			
25	1	0			

FIGURE 24 : TABLEAU des COMMANDES

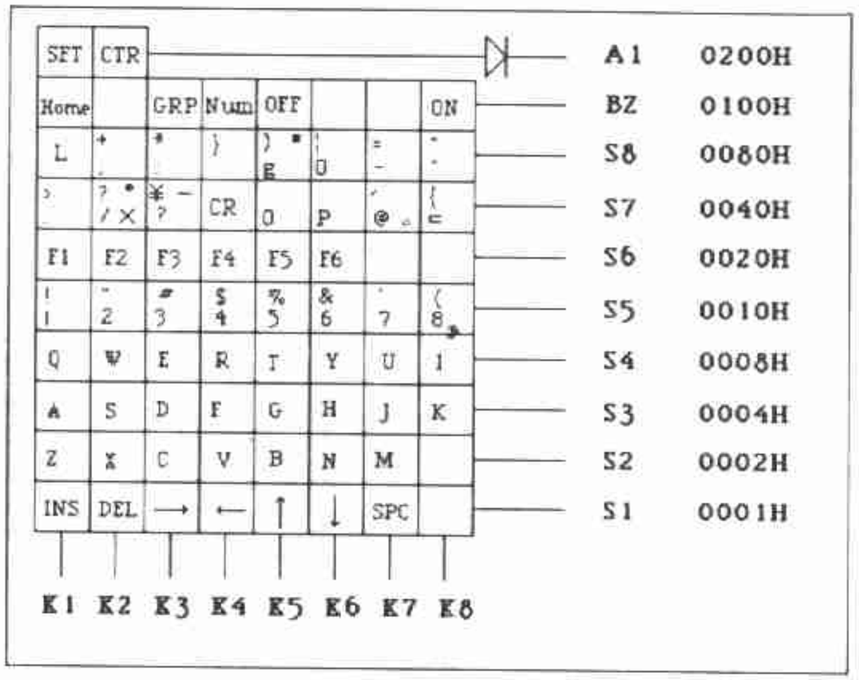


FIGURE 25 : MATRICE de CLAVIER

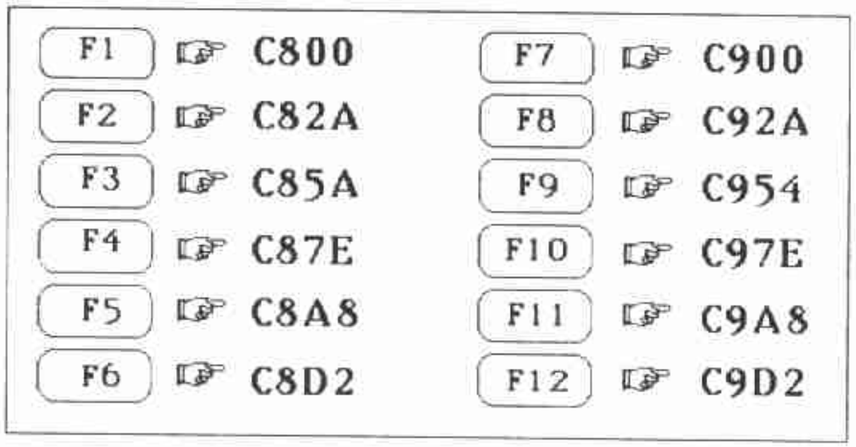


FIGURE 26 : ADRESSES des TOUCHES de FONCTION

7.6.2 Le secteur Mémoire du T6834 .

Afin de mémoriser l'état d'un écran , l'alarme , la date et les caractères graphiques , le sous-processeur dispose de **deux Kilo-octets de mémoire vive** divisés en plusieurs sections décrites ci-dessous .

_ **256 octets** sont réservés au "buffer du clavier" . Un buffer est un espace mémoire spécialement réservé au stockage de données . Dans ce cas précis , ce buffer emmagasine les caractères frappés au clavier et les ressort à allure constante dès que l'écran le permet . Le buffer est donc une sorte de barrage . On a donc **127 touches** stockées en permanence dans cette zone de 256 octets s'étendant des adresses **CC00h à CCFFh** .

_ **512 octets** sont réservés au programme de lancement . Il est constitué d'une routine que vous définissez vous-même et lancé automatiquement après un OFF/ON (si vous l'avez programmé au moyen de la commande START$) . Cette zone s'étend des adresses **CE00h à CFFFh** .

_ **512 octets** sont réservés aux caractères définis par l'utilisateur . Cette zone , s'étendant des adresses **CA00h à CBFFh** , est divisée en deux secteurs . En effet , on obtient les adresses des caractères correspondant aux codes ASCII 80h à 9Fh par la relation **CA00h + (ASCII - 80h) * 8** . Par contre , les adresses des caractères correspondant aux codes ASCII E0h à FFh sont données par la relation **CB00h + (ASCII - E0h) * 8** .

_ **512 octets** sont réservés aux touches de fonction définies par l'utilisateur . Cette zone s'étend des adresses **C800h à C9FFh** . Les adresses des douze touches de fonction sont décrites à la figure 26 .

_ **511 octets** sont occupés par la mémoire écran . Cette zone , s'étendant des adresses **E000h à E1FEh** stocke tout ce qui se trouve sur l'écran LCD du CANON X-07 .

D'autre part , nous vous donnons ci-dessous quelques **informations générales** sur cette MEV secondaire :

_ **C006h** : le BIT 0 contient l'indication ONSTATE . Le bit 6 contient l'état SLEEP . De plus , le BIT 7 contient le mode ALARM simulé .

_ **C00Ch** : cette adresse contient le mode de touches utilisé lors de la frappe au clavier . En effet , cette adresse peut contenir : 0 ---> NORMAL
1 ---> KANA
2 ---> GRAPHIQUE
3 ---> NUMERIQUE

_ **C01Ah** : l'alarme est validée si le bit 0 de cette adresse est égal à 1 .

_ **C040h à C04Eh** : tous les deux octets , les données du temps sont mémorisées . Les nombres stockés les uns après les autres représentent : les secondes , les minutes , les heures , le jour de la semaine , la date , le mois et l'année en cours .

_ **C050h à C05Eh** : de même , les données de l'alarme sont stockées entre ces deux adresses .

_ **C080h** : indicateur de curseur .

_ **C082h** : indicateur de touche définie par l'utilisateur .

_ **C084h** : compteur de buzzer . Le buzzer cesse de retentir lorsque l'indicateur de report est posé .

_ **C092h** : compteur de la minuterie d'alimentation .

_ **C096h** : fin du défilement de l'affichage .

_ **C09Eh** : début de défilement de l'affichage .

Après avoir vu toutes les commandes du T6834 et ausculté sa mémoire intime , nous allons apprendre à l'utiliser ...

7.6.3 L' UTILISATION du T6834 .

Eh oui !! C'est bien joli d'avoir 70 commandes mais il faudrait savoir les utiliser intelligemment afin de tirer parti d'un processeur puissant autorisant des foules de choses comme vous le verrez bientôt ...

Nous avons plusieurs possibilités pour se servir du T6834 . En effet , le BASIC du CANON utilise fréquemment le T6834 : il lui faut donc des routines toutes faites en ROM pour qu'il puisse communiquer aisément avec ce circuit .

Il existe deux routines principales se chargeant de ces transmissions . Leur adresse respective est **E428h et C92Fh** .

La routine **E428h** sert à envoyer un ordre précis au T6834 si l'on a aucun paramètre et que l'on attend pas de réponse . Il est impératif de placer le numéro de la commande dans le registre A et d'appeler ensuite la routine par un CALL $E428 . Attention !! En retour , les registres A , F et L seront modifiés .

Ainsi , la petite routine suivante fera clignoter l'écran du X-07 :

```
    LD B.$FF
#1  LD B.$2B
    CALL $E428
    LD A.$2C
    CALL $E428
    DJNZ #1
    RET
```

La deuxième routine d'adresse **C92Fh** est beaucoup plus puissante . En effet , elle permet le passage de paramètres et utilise six registres du NSC 800 de la façon suivante :

_ Dans **A** : numéro de la commande .
_ Dans **HL** : adresse des opérateurs ou paramètres .
_ Dans **B** : nombre de paramètres .
_ Dans **C** : nombre de réponses arrivées .
_ Dans **D** : pointeur du tampon qui stocke les données (En général , $26E) .

Le problème est qu'au retour , **les registres AF , BC , DE et HL seront modifiés** . Immobiliser autant de registres devient très vite embarrassant car le NSC 800 ne dispose pas de 200 registres !!

Nous avons résolu ce problème en développant **une petite routine** qui n'utilise en tout et pour-tout que **le registre E** !! De plus , seuls **les registres A et C seront modifiés** .

Cette routine permet d'envoyer le code d'une commande au T6834 ainsi que des paramètres et d'obtenir des réponses . L'utilisation de cette routine est donc très souple et vous permet d'économiser vos précieux registres .

Voici le listing de cette routine ainsi que divers commentaires . De plus , vous allez avoir droit à une petite démonstration graphique utilisant quelques possibilités du sous-processeur . Notez la vitesse de traitement : vous vous apercevrez que travailler en direct avec le T6834 n'est pas une chose vaine !!

Dans le listing joint , notons qu'au retour , aucun registre n'est modifié car ils sont sauvegardés et récupérés grâce aux instructions PUSH et POP . De plus , la routine **C9C0h** vérifie que le T6834 est prêt à recevoir des informations : elle ne rend la main au programme que s'il est prêt ...

Vous pouvez vous reporter au programme d'inversion VIDEO LCD illustrant lui aussi les possibilités du T6834 , présent dans la troisième partie .

MNEMONIQUES	COMMENTAIRES
PUSH AF	Sauvegarde du registre AF
PUSH DE	Sauvegarde du registre DE
PUSH BC	Sauvegarde du registre BC
LD C.$F1	C = Numéro du port de sortie
CALL $C0C9	Sous-processeur PRET ?
LD A.($26C)	Chargement du registre A
OR $80	Comparaison de A avec le code $80
OUT ($F0).A	Sortie de A sur le port $F0
OUT (C).E	Sortie de E sur le port $F1
LD A.$2	Chargement de A avec le chiffre 2
OUT ($F5).A	Sortie de A sur le port $F5
POP BC	Récupération du registre BC
POP DE	Récupération du registre DE
POP HL	Récupération du registre HL
RET	RETOUR AU BASIC

UNE PETITE DEMONSTRATION ...

```
5  REM programme BASIC implantant une routine en LANGAGE
   MACHINE contenant une démonstration ...
10 CLS : PRINT "Je charge les codes ... Un instant , SVP !"
20 FOR X = &H1C00 TO &H1C3E
30 READ A$ : POKE X,VAL ("&H" + A$)
40 NEXT X
45 REM données de la routine LANGAGE MACHINE
50 DATA CD , 9E , CE , 1E , 15 , CD , 25 , 1C
60 DATA 1E , 3C , CD , 25 , 1C , 1E , 0F , CD
70 DATA 25 , 1C , 1E , 0A , CD , 25 , 1C , 06
80 DATA FF , 1E , 2B , CD , 25 , 1C , 1E , 2C
90 DATA CD , 25 , 1C , 10 , F4 , F5 , C5 , D5
92 DATA 0E , F1 , CD , C0 , C9 , 3A , 6C , 02
95 DATA F6 , 80 , D3 , F0 , ED , 59 , 3E , 02
98 DATA D3 , F5 , D1 , C1 , F1 , C9 , 00 , 00
```


BASIC & ZONE SYSTEME

Ce chapitre va être consacré à l'étude de trois secteurs importants du CANON X-07 : **le codage en mémoire du BASIC , les points d'entrée en mémoire morte des fonctions BASIC et la zone de travail système** .

En effet , pour pouvoir programmer efficacement en langage machine , il vous faut connaître la structure interne du BASIC . Les fonctions BASIC font très souvent appel à des routines préprogrammées qui vous serviront dans vos logiciels écrits en ASSEMBLEUR .

8.1 La MEMOIRE VIVE .

Nous allons étudier le codage du langage BASIC MICROSOFT en mémoire vive . En effet , vous vous demandez sûrement comment sont stockées les fonctions et les instructions du BASIC après que avoir tapé sur la touche "RETURN" ...

La programmation en BASIC obéit à des règles très strictes . Tout est structuré impeccablement pour que le NSC 800 puisse exécuter n'importe quelle ligne du programme quand vous le lui demandez .

L'adresse de départ des programmes BASIC est **1363 en décimal** . De l'adresse 0 à 1362 se trouve **la zone système** que nous étudierons en détail au paragraphe 8.3 .

Chaque fonction BASIC (PRINT , GOTO , THEN ...) possède un code distinctif compris entre **80h et FEh** . Chaque ligne est séparée d'une autre par **le code 0** . Nous allons "PEEKER" ensemble à partir de l'adresse 1363 après avoir entré le petit programme suivant :

```
1 REM **
200 CLS : A$="C7"
9999 PRINT"END","C7":END
```

ADRESSES	CODE	INSTRUCTIONS	COMMENTAIRES
1363	92	INDICATEUR	Cet indicateur se trouve au
1364	05		début de chaque ligne et indique l'adresse de la prochaine ligne . (Ici : 92+256*5 = 1372)
1365	01	N° de LIGNE	Codé sur 2 octets . Ici , on a donc
1366	00		1+256*0 = 1 = ligne 1 .
1367	142	REM	Code de la fonction BASIC REM .
1368	32	ESPACE	Code ASCII de l'espace .

ADRESSES	CODE	INSTRUCTIONS	COMMENTAIRES
1369	42	*	Code ASCII de l'étoile .
1370	42	*	Code ASCII de l'étoile .
1371	00	Fin de LIGNE	A chaque fin de ligne , un 00 .
1372	106	INDICATEUR	Ligne suivante:106+256*5=1386.
1373	05		
1374	200	N° de LIGNE	Ici , on a 200+256*0=200
1375	00		
1376	166	CLS	Code de la fonction BASIC CLS .
1377	58	:	Code du séparateur BASIC .
1378	65	A	
1379	36	$	
1380	221	=	Codage d'une déclaration de
1381	34	"	variable alphanumérique .
1382	67	C	
1383	55	7	
1384	34	"	
1385	00	Fin de LIGNE	
1386	124	INDICATEUR	Ligne suivante:124+256*5=1404.
1387	05		
1388	15	N° de LIGNE	Ici , on a 15+256*39=9999 .
1389	39		
1390	159	PRINT	Code de la fonction BASIC PRINT
1391	34	"	Code ASCII des guillemets .
1392	69	E	Code ASCII de la lettre "E" .
1393	78	N	Code ASCII de la lettre "N" .
1394	68	D	Code ASCII de la lettre "D" .
1395	34	"	Code ASCII des guillemets .
1396	44	,	Code ASCII de la virgule .
1397	34	"	Code ASCII des guillemets .
1398	67	C	Code ASCII de la lettre "C" .
1399	55	7	Code ASCII du chiffre "7" .
1400	34	"	Code ASCII des guillemets .
1401	58	:	Code ASCII du séparateur .
1402	128	END	Code de la fonction BASIC END .
1403	00	Fin de LIGNE	

Vous pouvez remarquer la simplicité de codage du BASIC en mémoire vive . Dans le tableau de la figure 27 , **les 256 codes ASCII disponibles** vous sont présentés . Vous noterez que les fonctions BASIC sont codées au méme endroit que les caractères graphiques .

En ce qui concerne le codage des variables en mémoire , vous pouvez vous reportez aux manuels bleu et jaune du CANON X-07 et plus particulièrement à la fonction VARPTR .

FIGURE 27 :

TABLEAU des CODES

0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
		SPC	0	@	P	·	p	END	ELSE	CONT	FSET	STRING$	STEP	INT	FIX
	DC1	!	1	A	Q	a	q	FOR	TR	LIST	PAINT	INSTR	+ F M	ABS	LEN
	DC2	"	2	B	R	b	r	NEXT	MOTOR	LLIST	LOAD	INKEY$	- F M	FRE	HEX$
		#	3	C	S	c	s	DATA	DEFSTR	CLEAR	SAVE	INP	* F M	POS	STR$
		$	4	D	T	d	t	INPUT	DEFINT	CIRCLE	INIT	VARPTR	/ F M	SQR	VAL
		%	5	E	U	e	u	DIM	DEFSNG	CONSOLE	ERASE	USR	- F M	RND	ASC
		&	6	F	V	f	v	READ	DEF DBL	CLS	BEEP	SNS	AND	LOG	CHR$
BELL		'	7	G	W	g	w	LET	LINE	COLOR	CLOAD	ALM$	OR	EXP	TKEY
BS		(	8	H	X	h	x	GOTO	ERROR	EXEC	CSAVE	DATE$	XOR	COS	LEFT$
		)	9	I	Y	i	y	RUN	RESUME	LOCATE	NEW	TIME$	EQV	SIN	RIGHT$
LF		*	:	J	Z	j	z	IF	OUT	PSET	TAB	START$	MOD	TAN	MID$
LU		+	;	K	[	k	{	RESTORE	ON	PRESET	TO	FONT$¥	F M	ATN	CSRLIN
CLS	→	,	<	L	¥	l	|	GOSUB	DEFFN	OFF	FN	KEY$	> F M	PEEK	STICK
CR	←	-	=	M	]	m	}	RETURN	LPRINT	SLEEP	USING	SCREEN	= F M	CINT	STRIG
SO	↑	.	>	N	^	n	~	REM	POKE	DIR	ERL	THEN	< F M	CSNG	POINT
SI	↓	/	?	O	_	o	←	STOP	PRINT	DELETE	ERR	NOT	SGN	CDBL	

Voici quelques détails sur les abréviations du précédent tableau :

- **BELL** : sonnerie .
- **BS** : BACK SPACE .
- **LF** : LINE FEED (Déroulement d'une ligne) .
- **LU** : LINE UP (Retour en arrière d'une ligne) .
- **DC1** : DEVICE CONTROL 1 (Passage Texte/Graphique sur la X-710) .
- **DC2** : DEVICE CONTROL 2 (Passage Graphique/Texte sur la X-710) .
- **SI** : SHIFT IN (Utilisation de caractères pour la X-710 provenant d'une ROM EXTERNE) .
- **SO** : SHIFT ON (Utilisation de caractères pour la X-710 provenant de la ROM INTERNE) .
- **F.M** : Fonction mathématique .

8.2 Les POINTS D' ENTREE en MEMOIRE MORTE .

Nous ne vous apprendrons pas que le BASIC MICROSOFT du CANON X-07 est très puissant . Comme la puissance se paie , il a fallu **20 Kilo-octets de mémoire morte** (appelé aussi ROM comme Read Only Memory ou MEM) pour le stocker complètement .

L'interpréteur BASIC se trouve entre les adresses **B000h et FFFFh** , c'est à dire complètement en haut de la mémoire étant donné que l'adresse FFFFh représente la dernière adresse disponible .

Nous allons vous donner tous les points d'entrée en MEM des fonctions BASIC du X-07 . Avec ces adresses , vous pourrez fouiller la mémoire et récupérer toutes les routines utilisées par les fonctions BASIC (Voir figure 28) .

Mais si vous ne désirez pas fouiller la mémoire à grands coups de PEEK , reportez-vous au chapitre 9 ...

8.3 La ZONE de TRAVAIL du SYSTEME .

Appelé abusivement "zone système" , la zone de travail du système est constitué d'une **zone de mémoire vive** utilisée par le BASIC afin de faire des appels au système .

En ce qui concerne le CANON X-07 , la zone système occupe quelques **1363 octets de l'adresse 0000 à l'adresse 1362 en décimal** (0 à 552 en hexadécimal) .

MOTS CLES	CODES	ENTREE	MOTS CLES	CODES	ENTREE
ABS	E1	C9DC	LET	87	F685
ALM$	C7	D928	LINE	97	F7C6
AND	D6	FC68	LIST	A1	FE75
ASC	F5	D81A	LLIST	A2	FE6B
ATN	EB	B5AC	LOAD	B2	DF38
BEEP	B6	C2CD	LOCATE	A9	E212
CDBL	EF	CB90	LOG	E6	B609
CHR$	F6	D828	LOCATE	A9	E212
CINT	ED	CAE0	LOG	E6	B609
CIRCLE	A4	009C	LPRINT	9C	E9DE
CLEAR	A3	D1D0	MID$	FA	DC93
CLOAD	B7	DF30	MOTOR	92	E0B9
CLS	A6	CE9E	NEW	B9	D214
COLOR	A7	008A	NEXT	82	D30C
CONSOLE	A5	E243	NOT	CF	FC1A
CONT	A0	D178	OFF	AC	C5B1
COS	E8	B530	ON	9B	F6F1
CSAVE	B8	DEF9	OUT	9A	E98C
CSNG	EE	CB08	PAINT	B1	0099
CSRLIN	FB	CEAF	PEEK	EC	FC7A
DATA	83	F66A	POINT	FE	0093
DATE$	C8	D8CE	POKE	9E	FC80
DEFDBL	96	F557	POS	E3	FC8C
DEFFN	9D	FC96	PRESET	AB	0090
DEFINT	94	F551	PRINT	9F	EA05
DEFSNG	95	F554	PSET	AA	008D
DEFSTR	93	F54E	READ	86	F86B
DELETE	AF	E682	REM	8E	F666
DIM	85	B005	RESTORE	8B	D296
DIR	AE	E5E3	RESUME	99	F734
ELSE	90	F666	RETURN	8D	F63D
END	80	D2B0	RIGHT$	F9	DC8A
EQV	D9	FC72	RND	E5	B771
ERASE	B5	E3CF	RUN	89	F5B7
ERL	BE	FAC4	SAVE	B3	DEFE
ERR	BF	FAB6	SCREEN	CD	0087
ERROR	98	F78C	SGN	DF	C9F1
EXEC	A8	CEB8	SIN	E9	B549
EXP	E7	B6DD	SLEEP	AD	C59B
FIX	F0	CC14	SNS	C6	E961
FN	BC	FCB5	SQR	E4	B694
FONT$	CB	D739	START$	CA	DA99
FOR	81	F3F7	STEP	D0	F537
FRE	E2	DDD7	STICK	FC	D7E3
FSET	B0	BF56	STOP	8F	D2C0
GOSUB	8C	F610	STR$	F3	D53D
GOTO	88	F621	STRIG	FD	D7F8
HEX$	F2	D538	STRING$	C0	D835
IF	8A	F797	TAN	EA	B593
INIT	B4	E699	TIME$	C9	D868
INKEY$	C2	C103	TKEY	F7	D7CC
INP	C3	E931	TR	91	D1B7
INPUT	84	F82F	USR	C5	E9BB
INSTR	C1	DCD6	VAL	F4	DCB3
INT	E0	CC23	VARPTR	C4	E9A7
KEY$	CC	DA39			
LEFT$	F8	DC5A			
LEN	F1	D72D			

FIGURE 14 : POINTS D' ENTREE en MEM .

Pour les "techniciens du SOFT" , nous pouvons préciser que cette zone est divisée en **quatre parties** bien distinctes :

- De 0000 à 00AE : zone initialisée conventionnellement .
- De 00AE à 00B4 : zone initialisée lors de la mise sous tension .
- De 00B4 à 00BE : zone initialisée lorsque le BASIC et remis à zéro .
- De 00BE à 0552 : zone générale de travail du système .

Voici maintenant le **désassemblage complet de cette zone système** ainsi que les explications indispensables à ce long périple . Tous les chiffres sont donnés en Hexadécimal sauf contre-indication .

ADRESSES	INSTRUCTIONS	COMMENTAIRES
000-002	C9	RST 0 (Break point de la carte XP140F)
003		Nombres de caractères imprimés * taille
004	$50=80 décimal	Longueur d'une ligne sur la X-710
005		Taille des caractères sur la X-710
008-00A	JP F52F	RST 08 (Voir les fonctions systèmes , Chap. 9)
00C-00D		START de la ROM1
010-012	JP F537	RST 10 (Test caractère)
013-014		FIN de la ROM1
018-01A	JP C9C7	RST 18 (Emission de commande vers T6834)
020-022	JP EE45	RST 20 (Comparaison entre HL et DE)
024-025		START de la ROM2
025-027		FIN de la ROM2
028-02A	JP E88F	RST 28 (Emission de A vers dispositif ouvert)
02B		FLAG de BREAK
02C-02E	C9	INTERRUPT C (Clignotement curseur)
030-032	JP FC2F	RST 30 (Test du type de données)
034-036	JP C7A3	INTERRUPT B (Interruptions SERIE)
038-03A	JP E906	RST 38 (Calcul des adresses de tableaux)
03C-03E	JP C799	INTERRUPT A (Interruptions du T6834)
03F-041	JP CED6	Appel de GPRPUT (Imprimante parallèle)
042-044	JP E512	Emission du contenu de A vers KBD , CASI
045-046		Pointeur de la table pour RST 38
047-056		Table des HOCKS
057-059	JP E327	Minuterie automatique
05A-05C	JP C16C	RETOUR
05D	JP C1FA	
060	JP C181	
063	JP C219	
066	C9	NMI
069	JP C582	
06C	JP C0A1	LSET CURSEUR
06F	JP C210	
072	JP C231	Affiche le contenu de C à l'adresse écran

ADRESSES	INSTRUCTIONS	COMMENTAIRES
075	JP C250	SCREEN / Image miroir
078	JP C154	
07B	JP C138	
07E	JP C120	
081	JP C24B	TXT TAB
084	JP F1AA	SN ERROR
087	JP E403	SCREEN
08A	JP F1AA	Instruction COLOR PERITEL / Sinon SN ERROR
08D	JP CDF4	Instruction PSET
090	JP CDF7	Instruction PRESET
093	JP CE05	Instruction POINT
096	JP CE19	Instruction LINE
099	JP F1AA	Instruction PAINT PERITEL / Sinon SN ERROR
09C	JP CE32	Instruction CIRCLE
		La zone s'étendant des adresses 057 à 09E est le secteur de communication "afficheur LCD/ afficheur VIDEO PERITEL" .
09F	JP C1BE	Sortie CONSOLE (Implicite LCD)
0A2	JP C90A	Entrée CONSOLE (Implicite KBD)
0A5	JP C16D	
0A8	C9	ABORT HOCK (JP 7535 sur la carte XP 140F)
0AB	C9	ERREUR HOCK (JP 756C sur la carte XP 140F)
0B1		FLAG RESUME en cours
0B2-0B3		Pointeur TXT TAB (Début BASIC)
0B8-0B9		Adresse Curseur : X dans 0B8/Y dans 0B9
0BA		Longueur d'une ligne d'écran LCD
0BB		N° de la première ligne de roulement
0BC		N° de la dernière ligne de roulement
0BD		Nombre de lignes de roulement
0D5-1D4		Tampon d'entrée
1D8		FLAG variable (0 , elle existe / FF , à créer)
1D9		Type de données : 02 ---> %
		03 ---> $
		04 ---> !
		08 ---> #
1DB-1DC		Dernière ligne exécutée
1DD-1DE		Fin de la pile (Indicateur STKTOP)
1DF-1E0		Fin de la zone BASIC (Indicateur MEMSIZ)
1E1-1E2		Prochaine adresse disponible dans la table des manipulations de chaines
1E3-200		Table des manipulations de chaines
201		Longueur de la chaine (entre 0 et 255)
202-203		Adresse de la chaine

ADRESSES	INSTRUCTIONS	COMMENTAIRES
204-205		Prochaine adresse vide dans la zone de stockage des chaines (Indicateur FRETOP)
206-207		Pointeur du dernier code exécuté ou taille des des tableaux et FLAG de PRINT USING
208-209		D43A
20A-20B		Pile des boucles FOR-NEXT
20E		FLAG pour l'instruction FOR
20F		FLAG pour C977
210-211		Pointeur du début de la zone fichier RAM (Indicateur RAMSTR)
212-213		Pointeur de la fin de la zone fichier RAM (Indicateur RAMEND)
214-263		Image miroir de l'écran LCD
264-265		Sauvegarde du pointeur BASIC (Registre HL) si le CANON est éteint par SLEEP
266-267		Sauvegarde du pointeur de pile (Registre SP) si le CANON est éteint par SLEEP
268-26B		Mot-Clé (Jour , heure , minute , seconde) posé en RAMSTR-4 et RAMEND-1
26C		INTIMAG
26D		Nombre d'octets de réponse attendu par le T6834
26E-2C3		1er tampon de communication avec le T6834
2C4-2C5		Adresse pilote de sortie
2E8		Indicateur KBNTAB : table de INIT# (5 fois 8 octets) . Adresse + sauvegarde des registres HL , DE et BC
2F5		N° du dispositif ouvert
2F6-2F7		Adresse du pilote
2F8-2FD		FLAG DATE$ (0 si actif et 1 si ALM$ actif)
2FE		FLAG ALM$, TIME$, DATE$
2FF-304		FLAG pour CLOAD et CLOAD?
305-30A		Tampon pour la sauvegarde du nom de fichier cassette à trouver (6 octets)
30B		Tampon pour le nom de fichier lu sur la bande magnétique (SKIP ou FOUND)
30C		UCRIMAG
30D		RSERFLAG
30E		1 si INPUT en cours / 0 autrement
30F-310		FLAG pour l'instruction READ
311-312		Appel à l'adresse 0B2 par RUN
313-314		Numéro de ligne en cours d'exécution
317-318		Sauvegarde du registre SP pendant la phase active de l'interpréteur

ADRESSES	INSTRUCTIONS	COMMENTAIRES
319-31A		Numéro de la ligne où une erreur se produit
31B		Adresse du traitement d'erreur (ON ERROR)
31C-31D		FF si on a un ON ERROR / 0 autrement
31E-31F		Adresse du point décimal dans le tampon PRINT
320-321		Adresse de CONT
322-323		Adresse du dernier code exécuté si une erreur s'est produite
324-325		Table des variables (Indicateur VARTAB)
326-327		Table des tableaux (Indicateur ARRTAB)
328-329		Adresse du début de la zone libre (Indicateur STREND)
32A-343		Pointeur des DATA après un RESTORE n . Ces deux octets pointent sur le premier octet de chainage de la ligne n
344-345		Table des types de variables (26 lettres)
348-3AB		Indicateur PRMSTK
3AC-3AD		Pile (Appel par B07C)
		Indicateur PRMPRV
414		FLAG d'une recherche variable
415-416		Recherche adresse d'une variable
41E-44E		Tampon interne de l'ordre PRINT
455		Accumulateur / 44E exposants / 450-451 entiers
456		Octet de travail pour l'accumulateur
49F-4A6		Accumulateur 2
4A7		Octet de travail pour l'accumulateur 2
4AF-4C6		Tampon pour la commande RND
4C7		Sauvegarde de l'ancienne coordonnée X lors de l'instruction LINE-
4C8-4C9		Sauvegarde de l'ancienne coordonnée Y lors de l'instruction LINE-
4CA-4CB		Sauvegarde de la nouvelle coordonnée X lors de l'instruction LINE ( )-( )
4CC-4CD		Sauvegarde de la nouvelle coordonnée Y lors de l'instruction LINE ( )-( )
507-54D		2ème tampon de retour pour le T6834
54E-54F		FLAG carte XP 140F en place / Fonction LOG
		Zone de travail de la carte moniteur XP 140F (Début)

Après ce survol minutieux de la zone système du CANON , nous allons attaquer la partie primordiale de cette seconde partie : **les principales routines figées en ROM** ...

LES PRINCIPALES ROUTINES DE LA ROM DU X-07

Nous savons tous que ce neuvième chapitre est attendu avec une grande émotion . Effectivement , près de **89 adresses importantes** vont vous être dévoilées !!

Ces adresses sont divisées en **six grandes familles** :

- Les routines d'entrées/sorties
- Les routines de conversion
- Les routines arithmétiques
- Les routines mathématiques
- Les routines systèmes
- Les routines secondaires

Tous ces sous-programmes peuvent être appelés directement par le CANON X-07 dans ses programmes écrits en langage machine . Le format des routines exposées sera de la forme suivante :

Adresse Hexadécimale **Entrée** : registres à charger avec un paramètre .
Action : effet de la routine .
Sortie : information de retour (état d'un registre ...) .
Modifiés : information de modification (registre dont le contenu diffère avant et après l'appel) .

Après cette brève introduction , entrons vite dans le vif du sujet sinon vous allez défaillir !!

9.1 Les ROUTINES D' ENTREE / SORTIE .

9.1.1 Le BUZZER .

C2DF **E** : IY contient la tonalité , H la durée et L est égal à 0 .
A : fonction BEEP du BASIC (mêmes valeurs pour les paramètres) .
S : rien .
M : les registres AF , BC , DE , HL , IY , EI .

C331 **E** : IY contient la tonalité , HL la durée et C le drapeau de marche (Si C=FF , on déclenche le haut parleur / Si C=00 , on arrête) .
A : actionne le haut-parleur .
S : rien .
M : les registres AF , BC , DE , HL , EI .

9.1.2 Le CLAVIER .

C8C5 **E** : rien .
A : obtention de l'entrée clavier et affichage du curseur .
S : caractère frappé dans le registre A . Le drapeau C est égal à 1 s'il y a interruption .
M : le registre AF .

C90A **E** : rien .
A : saisie d'un octet au clavier (Simili INKEY$) .
S : caractère frappé dans le registre A . Le drapeau Z est égal à 1 s'il n'y a pas d'introduction .
M : le registre AF .

9.1.3 L' ECRAN LCD .

C1BE **E** : A contient le caractère à afficher .
A : écriture du contenu de A à l'écran et dans la mémoire image . La position du curseur est incrémentée . Les codes de contrôle sont admis .
S : rien .
M : aucun registre .

C231 **E** : H contient la coordonnée X du curseur , L contient la coordonnée Y du curseur , C contient le caractère à afficher .
A : le contenu de C est écrit à l'écran à la position définie par le LOCATE en cours . Le curseur reste immobile .
S : rien .
M : les registres AF , BC , DE .

CE9E **E** : rien .
A : efface l'écran (équivalent de la fonction CLS) .
S : rien .
M : le registre AF .

C18A **E** : charger le registre A avec le code ASCII du caractère à afficher .
A : affiche le contenu de A sur la position du curseur et fait avancer le curseur .
S : rien .
M : aucun registre .

C0A1 **E** : charger le registre A avec la valeur 0 ou 1 .
A : si le registre A est égal à 0 , le curseur est affiché sur l'écran LCD . Par contre , si le registre A est égal à 1 , le curseur disparait .
S : rien .
M : les registres AF , BC , DE .

9.1.4 L' IMPRIMANTE .

CFB7 E : rien .
A : initialise l'imprimante parallèle --> Mode texte , taille 2 , couleur 0.
S : rien .
M : les registres AF , BC , DE , HL .

CFB0 **E** : rien .
A : émission des codes de contrôle LF + CR vers l'imprimante parallèle.
S : rien .
M : les registres AF , BC , DE , HL .

CEF7 **E** : charger le registre A avec l'octet à émettre .
A : émission du contenu de A vers l'imprimante . Les indicateurs BUSY et BREAK sont testés .
S : rien .
M : les registres AF , BC , DE , HL .

CF11 **E** : charger le registre A avec un caractère .
A : envoie le contenu du registre A vers le point d'accès parallele .
S : aucun .
M : les registres AF , B , E .

9.1.5 Le sous-processeur T6834 .

E428 **E** : le registre A contient l'octet à émettre .
A : émission vers le T6834 d'une commande sans paramètre .
S : rien .
M : les registres AF , C .

C92F **E** : le registre A contient le code de la commande , HL contient le pointeur de la zone où sont stockés les paramètres , B contient le nombre de paramètres , C contient le nombre d'octets constituant la réponse , DE contient l'adresse du tampon retour (En général , la valeur 26E) .
A : émission d un ordre avec paramètres vers le T6834 .
S : réponse pointée par le registre DE .
M : les registres AF , BC , HL . De plus , le bit 7 de A doit être positionné si B est nul , c'est à dire s'il n'y a pas de paramètre .

C0BD **E** : rien .
A : vide les tampons clavier et sous-processeur .
S : rien .
M : le registre AF .

E348 **E** : le registre HL doit être chargé avec une adresse de MEV du T6834
A : lit un octet pointé par HL dans la MEV du T6834 .
S : le registre A contient les données revenant du T6834 .
M : les registres AF , BC , DE .

E334 **E** : le registre HL est chargé avec une adresse de MEV contenue dans la MEV du T6834 , le registre A contient les données à écrire dans la MEV du T6834 .
A : écriture d'un octet contenu dans A à l'adresse de la MEV du sous-processeur contenue dans HL .
S : rien .
M : les registres AF , DE , BC .

9.1.6 Les ROUTINES GENERALES .

C39D **E** : rien .
A : initialisation des points d'accès parallèle , série , optique , imprimante et cassette .
S : rien .
M : les registres AF , BC , DE .

CF2A **E** : rien .
A : contrôle des bits d'état . Attente du passage de la borne BUSY du point d'accès parallèle à l'état bas . Contrôle des bits OFF , BREAK , piles épuisées .
S : rien .
M : le registre AF .

9.2 Les ROUTINES de CONVERSION .

9.2.1 Les CONVERSIONS de TYPE .

CAE0 **E** : charger l'accumulateur avec la donnée à convertir .
A : convertit la donnée flottante en donnée entière (Fonction CINT) .
S : rien .
M : tous les registres sont modifiés .

CB08 **E** : charger l'accumulateur avec la donnée à convertir .
A : le contenu de l'accumulateur est converti en Simple Précision (Fonction CSNG) .
S : rien .
M : tous les registres sont modifiés .

CB1E **E** : charger la donnée entière dans l'accumulateur .
A : le contenu de l'accumulateur est converti en Simple Précision .
S : rien .
M : tous les registres sont modifiés .

CB90 **E** : charger l'accumulateur avec la donnée à convertir .
A : convertit le contenu de l'accumulateur en Double Précision (Fonction CDBL) .
S : rien .
M : tous les registres sont modifiés .

9.2.2 Les CONVERSIONS ASCII-BINAIRE .

F595 **E** : le registre HL doit pointer sur la chaine à convertir .
A : Conversion de la chaine ASCII en entier . La conversion prend fin au premier caractère non numérique .
S : le registre DE contient le résultat .

BA21 **E** : le registre HL pointe sur la chaine à convertir .
A : Conversion en flottant de la chaine . Le FLAG est ajusté .
S : le résultat se trouve dans l'accumulateur .

9.2.3 Les CONVERSIONS BINAIRE-ASCII .

BB98 **E** : on charge le registre HL avec la donnée à convertir .
A : Conversion de HL en ASCII . On peut se servir de BB98 pour afficher un registre à l'écran .
S : écriture à l'écran de la chaine ASCII .

BE5F **E** : L'entier doit se trouver dans l'accumulateur . De plus , le registre HL doit pointer sur le tampon de sortie .
A : Conversion du contenu entier de A en ASCII . B et C doivent contenir des valeurs supérieures à 6 .
S : la chaine se trouve dans le tampon .
M : tous les registres sont modifés .

BBAB **E** : charger l'accumulateur avec le nombre flottant .
B = nombre de digits à gauche du point décimal .
C = nombre de digits à droite du point décimal .
Si A=0 , conversion Binaire-ASCII pur .
Si A=X , Bit 7=1->Edition ; Bit 6=1
Bit 5=1->* en tête ; Bit 4=1->$ en tête
Bit 3=1->signe présent ; Bit 2=1->signe suivant valeur
Bit 0=1->notation exponentielle .
A : Conversion Flottant en ASCII .
S : Valeur dans le buffer pointé par HL (En général , 41E ...) .

9.3 Les ROUTINES ARITHMETIQUES .

9.3.1 Sur les ENTIERS .

CCC6 E : charger HL et DE avec les nombres entiers à additionner .
Le drapeau 1D9 est à 2 .
A : addition entière .
S : le résultat se trouve dans le registre HL . DE est préservé .
S'il y a OVERFLOW , le résultat se trouve dans l'accumulateur .
Le FLAG TYPE est actualisé à 8 .

CCBB E : charger HL et DE avec les nombres à soustraire (HL-DE) .
Le drapeau 1D9 est à 2
A : soustraction entière .
S : le résultat se trouve dans HL . DE est préservé .
S'il y a OVERFLOW , le résultat est dans A . FLAG TYPE à 8 .

CCE7 **E** : charger HL et DE avec les nombres à multiplier .
Le drapeau 1D9 est à 2 .
A : multiplication entière .
S : le produit se trouve dans HL . DE est préservé .
S'il y a OVERFLOW , le résultat se trouve dans A . FLAG TYPE à 8 .

FA79 **E** : charger DE et HL avec les nombres à diviser (DE/HL) .
A : division entière .
S : quotient en double précision dans l'accumulateur .

9.3.2 En SIMPLE PRECISION .

CDA9 **E** : charger un nombre dans A , l'autre dans BC et DE .
Le drapeau 1D9 est à 4 .
A : addition .
S : le résultat se trouve dans l'accumulateur .

CDB2 **E** : charger l'accumulateur avec le 1er nombre et BC,DE avec le 2ème .
A : soustraction (A-(BC,DE)) .
S : le résultat se trouve dans A .

CDB7 **E** : charger l'accumulateur avec le 1er nombre et BC,DE avec le 2ème .
A : multiplication (A*(BC,DE)) .
S : le résultat se trouve dans A .

CDC2 **E** : charger l'accumulateur avec le 2ème nombre et BC,DE avec le1er .
A : division ((BC,DE)/A) .
S : le résultat se trouve dans A .

9.3.3 En DOUBLE PRECISION .

B20E **E** : charger l'accumulateur 2 et l'accumulateur .
Le drapeau 1D9 est à 8 .
A : addition .
S : le résultat se trouve dans l'accumulateur .

B200 **E** : charger l'accumulateur 2 et l'accumulateur .
A : soustraction (Accum. - Accum. 2) .
S : le résultat se trouve dans A .

B376 **E** : charger l'accumulateur 2 et l'accumulateur .
A : multiplication .
S : le résultat se trouve dans A .

B42F **E** : charger l'accumulateur 2 et l'accumulateur .
A : division (Accum. / Accum. 2) .
S : le résultat se trouve dans A .

9.3.4 De COMPARAISONS .

CA7B **E** : charger BC,DE avec le 1er nombre et A avec le 2ème .
A : comparaison en Simple Précision .
S : Si (BC,DE) > (A) , A = -1 . Les () marquent le contenu du registre .
Si (BC,DE) < (A) , A = 1 .
Si (BC,DE) = (A) , A = 0 .

CAA5 **E** : charger HL et DE avec les deux nombres à comparer .
A : comparaison entière .
S : Si (DE) > (HL) , A = -1 .
Si (DE) < (HL) , A = 1 .
Si (DE) = (HL) , A = 0 .

CAD9 **E** : charger Accum. 2 et Accum .
A : comparaison en Double Precision .
S : Si (Accum. 2) > (Accum.) , A = -1 .
Si (Accum. 2) < (Accum.) , A = 1 .
Si (Accum. 2) = (Accum) , A = 0 .

C9FB **E** : charger l'accumulateur . Le drapeau 1D9 est à 8 .
A : donne le signe en Double Précision .
S : Si (A) > 0 , A = 1 .
Si (A) < 0 , A = -1 .
Si (A) = 0 , A = 0 .

9.4 Les ROUTINES MATHEMATIQUES .

C9DC E : charger l'accumulateur avec un nombre .
A : l'accumulateur est converti en son équivalent positif . (Fonction ABS) .
S : valeur positive dans A .

CC23 E : charger l'accumulateur avec un nombre .
A : donne la partie entière du nombre stocké dans A . (Fonction INT) .
S : la partie entière de l'accumulateur est laissée dans A . Le drapeau 1D9 est actualisé .

9.5 Les ROUTINES SYSTEMES .

9.5.1 Les ReSTarts .

RST 00 Correspond au "BREAK POINT" de la carte MONITEUR XP 140F . Un code C9 (RET) est implanté à l'adresse 0000 .

RST 08 La routine correspondante est implantée en F52F .
Cette commande compare deux symboles .
Comparaison entre l'octet pointé par HL et l'octet qui suit l'appel par RST 08 : _ Si concordance , retour à RST 08+2 et HL incrémenté .
_ Si désaccord , "SN ERROR" s'affiche .

RST 10 La routine correspondante est implantée en F537 .
Cette commande examine le symbole suivant . De plus , elle charge dans A le caractère pointé par le registre HL .
En sortie , HL est incrémenté et : C=1 si caractère alphanumérique .
C=0 si caractère alphabétique .

RST 18 La routine correspondante est implantée en C937 .
Cette instruction émet un ordre vers le T6834 .
A doit contenir le code de commande (Pas de paramètre) .

RST 20 La routine correspondante est implantée en EE45 .
Cette instruction compare DE avec HL .
_ Si HL < DE , C=1 .
_ Si HL > DE , C=0 .
_ Si HL = DE , Z=1 .

RST 28 La routine correspondante est implantée en E88F .
Cette instruction émet le contenu de A vers le dispositif ouvert , en général l'écran LCD . Tous les registres sont préservés .

RST 30 La routine correspondante est implantée en FC2F .
Cette instruction teste le type de données se trouvant dans A . On a :

1D9	TYPE	FLAGS	A
2	ENTIER	NC , Z , M	-1
3	CHAINE	Z , C , P	0
4	Simple PRE	NZ , C , P	1
8	Double PRE	NZ , NC , P	5

RST 38 La routine correspondante est implantée en E906 .
Cette commande calcule l'adresse des tableaux .

9.5.2 Les MOUVEMENTS de DONNEES .

CA0B E : charger l'accumulateur avec une donnée en Simple Précision .
A : empile (PUSH) le contenu de A .
Le drapeau 1D9 n'est pas testé .
S : la pile a augmenté d'une donnée .

CA1B E : charger les registres BC et DE avec des valeurs .
A : le contenu des registres BC et DE transitent vers le registre A .
S : l'accumulateur stocke la nouvelle valeur .

CA26 E : charger le registre A avec une valeur .
A : le contenu de A transite vers les registres BC et DE .
S : les registres BC et DE stockent la nouvelle valeur .

CA42 **E** : charger l'accumulateur avec une valeur et le registre HL avec l'adresse d'un tampon .
A : mouvement du contenu de A vers un tampon pointé par HL .
S : le tampon se remplit .

CA51 **E** : charger B avec le nombre d'octets à mouvoir . De plus , DE doit contenir l'adresse du tampon de départ et HL , l'adresse du tampon d'arrivée .
A : mouvement d'un groupe d'octets vers un tampon .
S : HL et DE croissent .

CA58 **E** : pareil que CA51 .
A : pareil que CA51 .
S : HL et DE décroissent .

CAEF **E** : charger HL avec une valeur .
A : le contenu de HL transite vers le registre A .
S : A prend la valeur de HL . Le drapeau 1D9 prend la valeur 2 .

CA5F **E** : charger l'accumulateur 2 avec une valeur .
A : le contenu de l'accumulateur 2 transite vers l'accumulateur .
S : le registre A prend la valeur de Accum. 2 .

CA67 **E** : charger l'accumulateur avec une valeur .
A : le contenu du registre A transite vers l'accumulateur 2 .
S : l'Accum. 2 prend la valeur du registre A .

9.6 Les ROUTINES SECONDAIRES .

C02D **E** : rien .
A : interrogation de BREAKFLAG (Octet situé en 2B : le bit 0 contient l'indicateur BREAK , le bit 1 l'indicateur "piles épuisées" , le bit 3 l'indicateur "pile carte épuisée" , le bit 6 l'indicateur d'arrêt de minuterie et le bit 7 l'indicateur OFF) .
S : Si le bit d'introduction (Bit 6) est à 1 , la commande C02D le met à 0 , vide le tampon de touches et met à 1 le bit d'indication Z .
Par contre , si le bit 6 est à 0 , la commande C02D envoie la commande au programme appelant , avec l'indicateur Z réglé .
M : aucun registre .

C5A6 **E** : rien .
A : controle les bits d'épuisement de la pile de la carte et des piles du X-07 (Controle des bits de BREAKFLAG) .
S : Si le bit de piles épuisées est à 1 , affichage de "LOW BATTERY" et retour au programme appelant , avec l'indicateur Z remis à 0 .
Si le bit de la pile de carte épuisée est à 1 , le message "CARD LOW BATTERY est renvoyé et on retourne au programme appelant avec l'indicateur Z remis à 0 .
Par contre , si les deux bits précédents sont à 1 , on retourne au programme appelant avec l'indicateur Z mis à 1 .
De plus , le registre A est chargé avec les dernières données BREAKFLAG .
M : aucun registre .

EE13 **E** : charger HL avec l'adresse de saut de tableau et C avec le décalage.
A : saut dans le tableau du vecteur .
S : HL change de valeur .
M : les registres AF , BC , DE , HL .

C557 **E** : rien .
A : controle du commutateur de prise (MEV/MEM) .
S : cette commande charge HL avec 4000h si le commutateur de prise est réglé sur la position MEV .
Sinon , HL est chargé avec la valeur 2000h (Position MEM) .
M : les registres AF et HL .

C5C3 **E** : rien .
A : coupe l'alimentation du CANON .
S : rien .
M : tous les registres .

1C 6F 76 15 C3C5

C5CD E : charger le registre A avec 1 ou 2 .
A : valide ou invalide le programme de lancement .
Si le registre A est égal à 1 , le programme de lancement est validé et le CANON X-07 est éteint .
Par contre , si le registre A est égal à 2 , le programme de lancement est invalidé et l'alimentation du X-07 est coupée .
S : rien .
M : aucun registre .

C5A6 E : rien .
A : controle de BREAKFLAG .
S : Si l'indicateur S est à 1 , tous les bits de BREAKFLAG sont remis à 0 et le CANON est éteint .
Si l'indicateur S est à 0 , la commande est renvoyée au programme principal .
M : aucun registre .

E327 E : rien .
A : initialisation de la minuterie d'alimentation de 15 minutes en remettant à 0 le bit d'arrêt de minuterie de BREAKFLAG .
S : rien .
M : les registres AF , B .

C62E E : charger HL avec l'adresse à interroger .
A : la commande C62E controle si l'adresse contenue dans HL est bien une adresse de MEV .
S : Si l'adresse est incluse dans la MEV , le drapeau Z est mis à 1 .
Sinon , le drapeau Z est mis à 0 .
M : les registres AF , B .

C59B E : rien .
A : introduction de l'état SLEEP OFF . SLEEP sauvegarde l'indicateur de piles actuel à l'adresse 226h et le contenu de HL à l'adresse 264h . Puis , l'état SLEEP OFF est introduit .
S : rien .
M : aucun registre .

E8E8 E : rien .
A : interrogation de BREAKFLAG .
S : l'indicateur Z est mis à 1 si un bit quelconque de BREAKFLAG est à 1 .
M : le registre AF .

F680 CONT général du système .

D5B0 Cette commande affiche un message sur l'afficheur LCD . Il suffit de charger le registre HL avec l'adresse de la chaine à afficher . La chaine doit être terminée par un zéro ou par le code 0D . Utilisation de la zone des chaines BASIC .

FEF7 Cette commande affiche également un message , comme ci-dessus , mais n'utilise pas la zone des chaines BASIC . La chaine doit être également terminée par un zéro .

EBF2 Cette instruction permet d'obtenir l'entrée clavier avec l'affichage du point d'interrogation . En sortie , le registre HL pointe sur l'adresse du tampon d'entrée -1 .

EBFF Cette commande a les mêmes spécifications que l'instruction EBF2 à part que le point d'interrogation n'est pas affiché .

F30D E : charger le registre DE avec le numéro de ligne à trouver .
A : recherche l'adresse d'une ligne écrite sous BASIC .
S : la ligne est trouvée quand les indicateurs C et Z sont égaux à 1 .
Alors , le registre BC contient l'adresse de la ligne en question .

FE5E E : charger le registre A avec une valeur .
A : cette commande évalue de façon entière (nombres < 255) le contenu du registre A (expression , constante ...) .
S : le résultat se trouve dans le registre DE et le poids faible dans A .
L'affichage de "FC ERROR" peut intervenir si A>255 .

FFCC Comme ci-dessus , la commande FFCC évalue de façon entière .
Le résultat se trouve dans les registres DE et A .

C528 A cette adresse , on arrive au niveau de l'affichage du message "COPY-RIGHT" .

C3C3 Initialisation générale . Départ à froid du X-07 .

FB4F Entrée d'une routine hexadécimale .

C4D1 Affichage de "CREATE SYSTEM ?" . Taper Y , "RETURN" pour continuer . Routine très pratique pour effacer complètement une carte mémoire .

org 7000:6C6F7665D1C6 : reset à l'allumage

F23D Mode "attente curseur" .

F1AA Affichage du message "SN ERROR" .

F1C7 Charger le registre E avec le numéro de l'erreur à créer . Le saut à cette adresse permet d'afficher l'erreur désirée .

Après cet inventaire qui réjouira plus d'un programmeur , nous allons aborder le dernier chapitre de cette seconde partie . Il est consacré **aux trois périphériques** les plus importants du CANON X-07 : la carte mémoire , l'imprimante X-710 et l'interface PERITELEVISION X-720 .

TROIS PERIPHERIQUES

Bien que les informations sur les périphériques du CANON X-07 soient assez rares , nous allons quand même vous parler des trois périphériques les plus utilisés : **la carte mémoire , la X-710 et la X-720** .

10.1 Les CARTES MEMOIRE .

Du point de vue logiciel , bien peu de choses restent à dire , l'interface ayant déja été abordé dans les chapitres précédents . Deux types de cartes sont disponibles : les cartes comprenant uniquement de la **RAM** (4 ou 8 Kilo-octets) et les cartes comprenant un programme en **ROM** (en général , 8 Ko) et 4 Kilo-octets de **RAM** (dont 3 utilisateur car un kilo-octet est réservé par la carte comme zone de travail) .

Pour les techniciens , voici le détail du brochage de ces cartes :

PATTE	SIGNAL	PATTE	SIGNAL
1	VCC1 (Alimentation	19	A12
2	VCC2 du X-07)	20	A11
3	NC	21	A10
4	V+ (Détection LOW BAT.)	22	A9
5	NC	23	A8
6	Accès RAM (RAM ON)	24	A7
7	Niveau bas sélection RAM	25	A6
8	WR	26	A5
9	RD	27	A4
10	Niveau bas sélection ROM	28	A3
11	D7	29	A2
12	D6	30	A1
13	D5	31	A0
14	D4	32	Masse
15	D3	33	Masse
16	D2		
17	D1		
18	D0		

10.2 L' IMPRIMANTE GRAPHIQUE X-710 .

Elle possède son propre micro-processeur **4 bits** ainsi qu'**une ROM de 4096 * 10 bits et une RAM de 256 * 4 bits** . Le tout fonctionne grâce à une horloge interne battant à quelques 600 Khz .

Les caractères imprimés correspondent aux codes ASCII selon la table décrite dans les manuels d'utilisation du CANON .

Les informations de déplacement du stylo sont interprétés de la façon suivante :

Bit 7 --> information de fin (1 , fin de déplacement / 0 , déplacement) .
Bits 4 à 6 --> taille du mouvement .
Bit 3 --> s'il est égal à 0 , le stylo est levé . S'il est égal à 1 , le stylo trace .
Bits 0 à 2 --> direction du mouvement .

A titre purement indicatif , nous vous exposons le plan de l'imprimante X-710 . En effet , les inconditionnels du "fer à souder" trouveront dans l'exploitation de ce plan et du brochage joint , la possibilité de rajouter une ROM contenant d'autres caractères ... Nous vous souhaitons bonne chance !!

Les deux figures présentes dans les pages suivantes vont relancer les ventes de fer à souder dans les prochaines semaines ... Merci aux auteurs !!

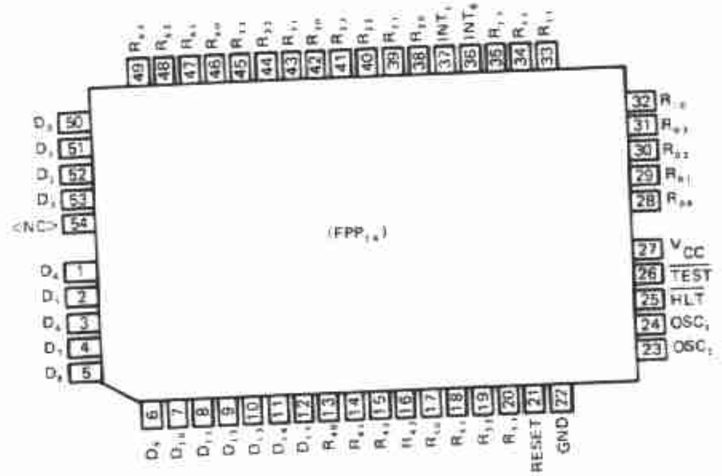
10.3 L' INTERFACE VIDEO X-720 .

La X-720 contient **6 Kilo-octets de mémoire vidéo** contenus entre les adresses **800h et 97FFh** . Cette mémoire est totalement utilisée par les modes SCREEN 5 et 6 .

Dans les modes "Basse résolution" , deux pages d'écran sont accessibles permettant des sauts de page à grande vitesse .

De plus , cette interface contient **4 Kilo-octets de ROM** , des adresses **4000h à B000h** . Elle modifie certaines adresses de la zone système selon la même méthode que les cartes . Par exemple , la carte XP 140F modifie plusieurs adresses de retour par des sauts (Voir paragraphe 8.3) . De même , la X-720 , en modifiant certaines adresses , obtient diverses fonctions BASIC nécessaires à son fonctionnement : SCREEN , PAINT , COLOR . Notons que les instructions comme LINE , PSET , PRESET , CIRCLE sont étendues .

A titre indicatif , nous désirons répondre aux personnes qui recherchent l'utilité du connecteur placé au dos de la X-720 . Normalement , cette prise devrait servir soit à des cartouches ROM , soit à des extensions de mémoire vive . Mais aucune extension n'a encore été commercialisée à ce jour ...



Pin contents

Pin No.	Designation	Input	Output	Form of terminal	Function
1	D4	○		A	SCS (Scale Set) L: Scale 1 after reset; H: Scale 0 after reset
2	D5	○		A	CRM (Carriage Return Mode) L: CR only; H: CR + LF
3	D6		○	A	EXEN (Rom Enable)
4	D7	○		B	Paper feed — Switch input from outside
5	D8	○		B	Back feed — Switch input from outside
6	D9	○		B	Color change — Switch input from outside
7	D10	○		B	Pen change — Switch input from outside
8	D11	○		A	PWS (Paper Width Select) L: 58 mm; H: 114 mm
9	D12	○		A	CNS (Color Number Select) L: 1 color; H: 4 colors
10	D13		○	C	(Halt Enable) HLTEN (Possitive logic)
11	D14		○	C	STR (Strobe Latch F/F reset)
12	D15		○	C	BSR (Busy Latch F/F reset)
13	R40	○		A	D_0 — Parallel data input — Character code input from outside
14	R41	○		A	D_1
15	R42	○		A	D_2
16	R43	○		A	D_3
17	R50	○		A	D_4 — Parallel data input
18	R51	○		A	D_5
19	R52	○		A	D_6
20	R53	○		A	D_7

Pin No.	Designation	Input	Output	Form of terminal	Function
21	RESET				Reset
22	GND				Negative power supply
23	OSC_1				Oscillator connection terminal
24	OSC_2				
25	HLT	O			Halt signal
26	TEST	O			Connection to Vcc
27	Vcc				Positive power supply
28	R00	O		A	EXD_0 — External ROM Data input. Connected to VCC when the External ROM is not used
29	R01	O		A	EXD_1
30	R02	O		A	EXD_2
31	R03	O		A	EXD_3
32	R10	O		A	EXD_4
33	R11	O		A	EXD_5
34	R12	O		A	EXD_6
35	R13	O		A	EXD_7
36	INT_0	O		A	Strobe input (initial edge trigger)
37	INT_1	O		A	Connected to GND
38	R20		O	C	RA_0 — External ROM Address
39	R21		O	C	RA_1
40	R22		O	C	RA_2
41	R23		O	C	RA_3
42	R30		O	A	$\overline{XD}$ — X direction motor drive signal
43	R31		O	A	$\overline{XC}$
44	R32		O	A	$\overline{XB}$
45	R33		O	A	$\overline{XA}$
46	R60		O	A	$\overline{YD}$ — Y direction motor drive signal
47	R61		O	A	$\overline{YC}$
48	R62		O	A	$\overline{YB}$
49	R63		O	A	$\overline{YA}$
50	D0		O	A	$\overline{PEN\ UP}$
51	D1		O	A	$\overline{PEN\ DOWN}$
52	D2	O		B	$\overline{COLOR\ DTECT}$
53	D3		O	C	$\overline{BAS}$ External (Rom Address Strobe)
54	NC				

Note: Form of terminals
A. Open drain
B. Pull-up MOS attached
C. CMOS output

FIGURE 29 : BROCHAGE de la PUCE

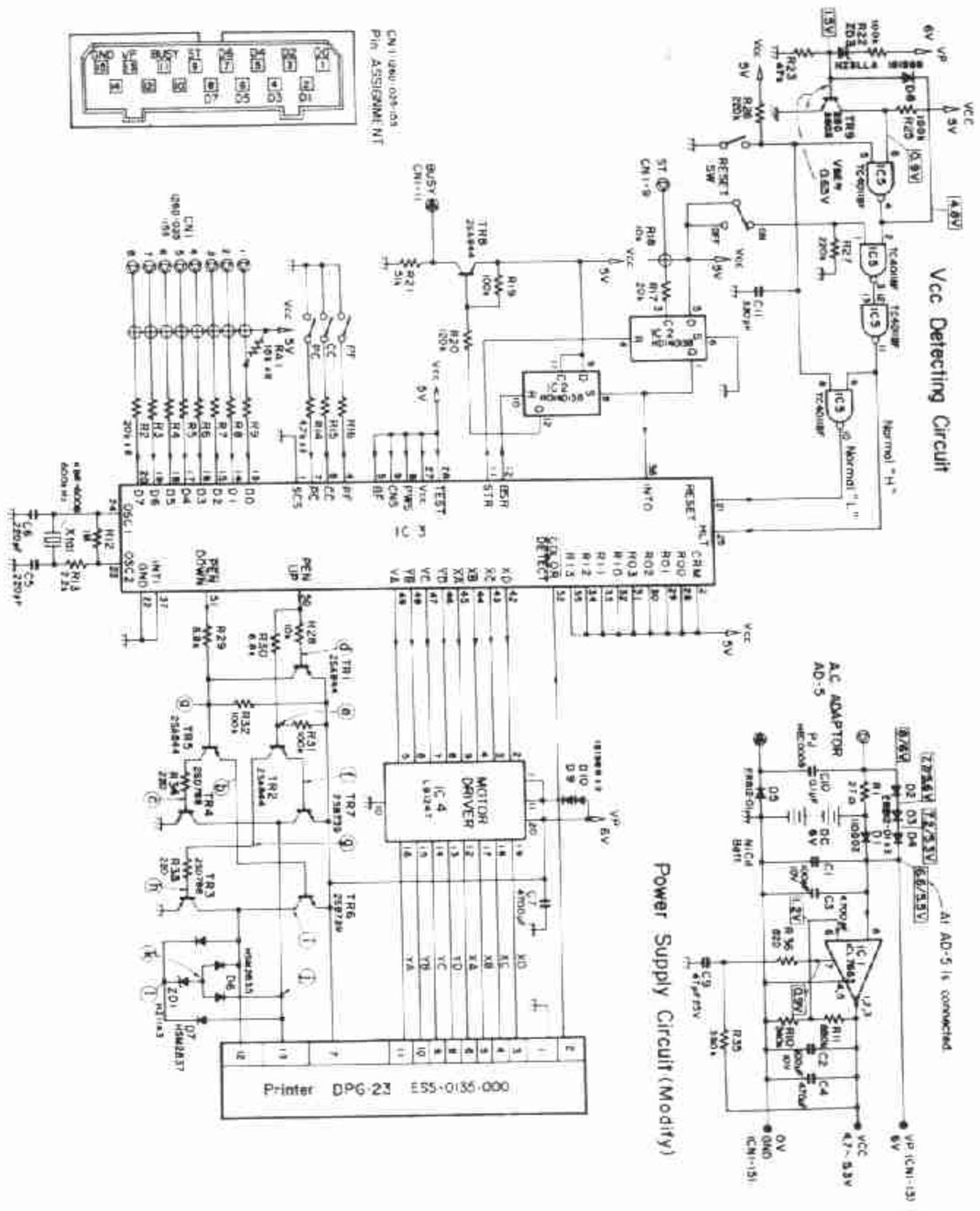
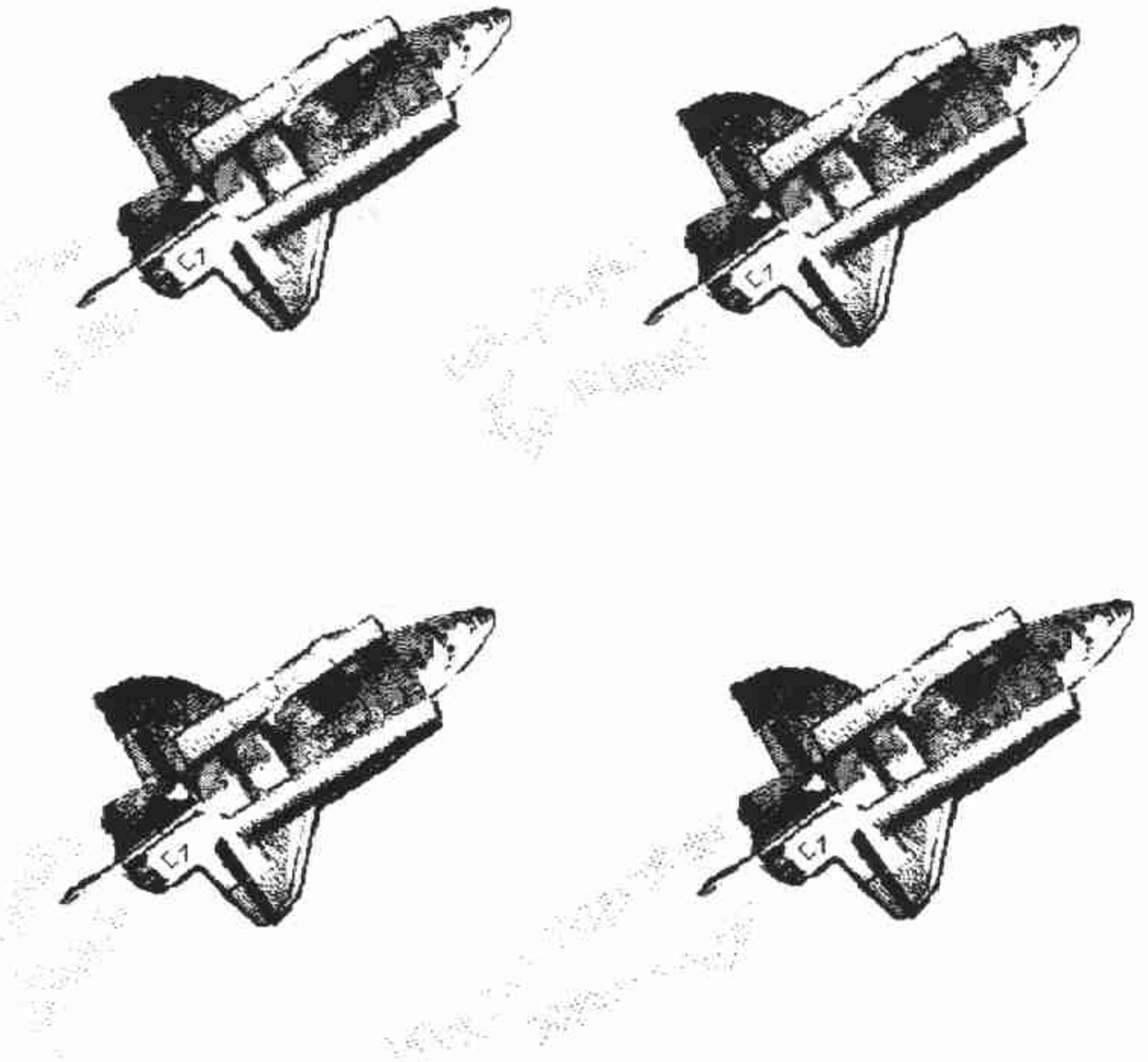


FIGURE 30 : PLAN de la X-710

Enfin !! Le mur impénétrable protégeant les entrailles du CANON X-07 a été percé ... Désormais , cette ouverture va vous permettre de progresser dans la programmation de votre machine préférée .

Il est vrai que les mystères du X-07 sont innombrables et que nous ne pourront jamais dévoiler toutes ses possibilités . Néanmoins , ces deux premières parties doivent servir de tremplin d'envol à votre essor vers le langage machine ...

Ne vous inquiétez pas , nous n'allons pas vous quitter si tôt : un lot d'applications va vous permettre de visualiser tous les concepts abordés .



VOLS SPATIAUX VERS LE SECTEUR "APPLICATIONS"

DEPECHEZ-VOUS !!!

3ème PARTIE :

APPLICATIONS

AFFICHAGE

Ce programme permet d'afficher **un motif de 8*8 points** en l'écrivant directement dans **la mémoire LCD** du sous-processeur T6834 (Voir le paragraphe 7.7 pour plus de précisions).

L'avantage de cette routine est constitué par le fait que nous ne sommes plus limités par des caractères définis par 6*8 points.

Le principe en est relativement simple. En effet, en RAM, le plus petit point adressable est constitué par une "ligne" de 8 points dans laquelle le chiffre 1 représente un point allumé et le chiffre 0 un point éteint.

On définira donc la forme désirée par **une suite de 8 octets où tous les chiffres 1 représenteront des points allumés**. Ensuite, nous irons écrire tous ces points les uns après les autres dans **la RAM du sous-processeur**.

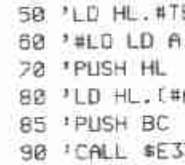
En fait, nous utilisons le même principe que l'instruction BASIC FONT$.

Reportez-vous aux listings de la figure 31 afin de bien comprendre le déroulement du logiciel. Le programme BASIC chargeant les codes est directement utilisable.

Longueur de la routine : **27 octets**.
Implantation de la routine : **de 1B00h à 1B1Ah**.

```
10 '[
20 'ORG $1B00
30 '* AFFICHAGE
40 'LD B.$08 :*NB DE LIGNES
50 'LD HL.#TB:*AD. DE LA TABLE
60 '#LO LD A.(HL)
70 'PUSH HL
80 'LD HL.(#AD)
85 'PUSH BC
90 'CALL $E334 :*ECRITURE EN RAM
95 'POP BC
100 'LD DE.$0010
110 'ADD HL.DE
120 'LD (#AD).HL
125 'POP HL
126 'INC HL
130 'DJNZ #LO
135 'RET
140 '#TB DEFB $FF,81,81,81,81,81,81,FF
150 '#AD DEFB $00,E0
160 ']
```

```
10 CLS:PRINT"un instant !"
20 FORI=&H1B00 TO &H1B24
30 READ A$:POKE I,VAL("&H"+A$)
40 NEXT I
50 DATA 06,08,21,1B,1B,7E,E5,2A,23,1B,C5
,CD,34,E3,C1,11,10,00,19,22,23,1B
60 DATA E1,23,10,EB,C9,FF,81,81,81,81,81
,81,FF,00,E0
```



Codage graphique	Codage binaire	Codage hexa.
	11111111	FF
	10000001	81
	10000001	81
	10000001	81
	10000001	81
	10000001	81
	10000001	81
	11111111	FF

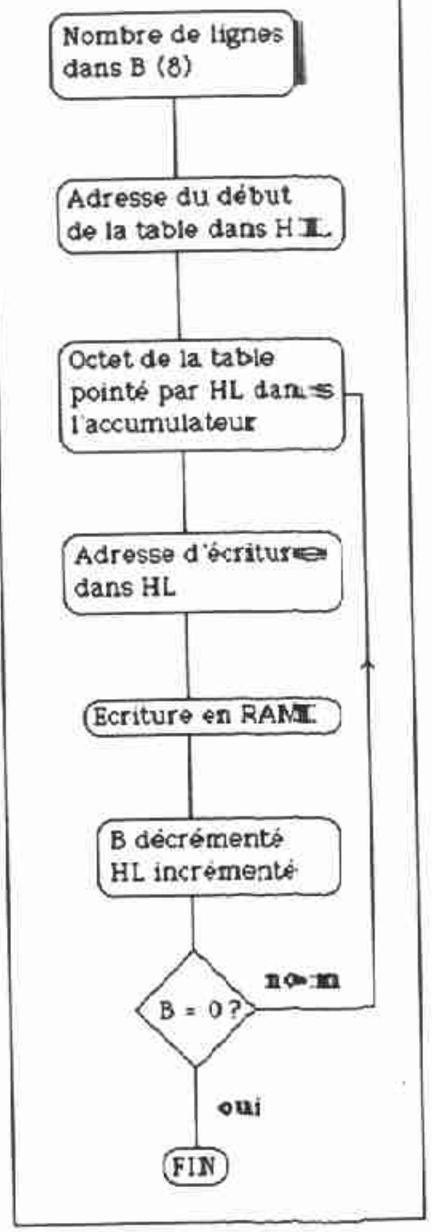


FIGURE 31 : AFFICHAGE

INVERSION DE L'AFFICHEUR VIDEO

Le but de cette routine est **d'inverser octet par octet la mémoire d'écran VIDEO du X-07 dans les modes SCREEN 5 et 6** .

On obtient ainsi une inversion des couleurs de l'affichage **quasi instantanée** .

Le principe du logiciel est basé sur une petite astuce logique ... Effectivement , nous utilisons l'instruction **XOR** dont la table de vérité est décrite dans la première partie de cet ouvrage .

En chargeant le registre B uniquement avec des chiffres 1 , on effectue le XOR logique avec le registre A et nous obtenons alors :

1 XOR 1 = 0 **et** **1 XOR 0 = 1**

Donc , en partant de l'adresse minimale de la mémoire VIDEO (Rappel : cette V.RAM s'étend des adresses **8000h à 97FFh**) , nous pouvons inverser octet par octet chaque parcelle de cette mémoire en suivant le principe précédemment décrit ... Qui a dit que l'interface VIDEO était lente ? ...

Longueur de la routine : **22 octets** .
Implantation de la routine : **de 1C00h à 1C15h** .

```
10 '[
15 '*INVERSE VIDEO
20 'ORG $1C00
30 'LD B,$FF
40 'LD HL,$8000 :*DEBUT DE LA VRAM
50 '#1E LD A,(HL)
60 'XOR B
70 'LD (HL),A
80 'LD A,H
90 'CP $97 :*OCTET FORT DE L'AD.DE FIN
100 'JR Z,#FI
110 '#2E INC HL
120 'JR #1E
130 '#FI LD A,L
140 'CP $FF:*OCTET FAIBLE DE L'AD.DE FIN
150 'RET Z
160 'JR #2E
170 ']

1C00 06FF       LD    B,FF
1C02 210080     LD    HL,8000
1C05 7E         LD    A,(HL)
1C06 A8         XOR   B
1C07 77         LD    (HL),A
1C08 7C         LD    A,H
1C09 FE97       CP    97
1C0B 2803       JR    Z,1C10
1C0D 23         INC   HL
1C0E 18F5       JR    1C05
1C10 7D         LD    A,L
1C11 FEFF       CP    FF
1C13 C8         RET   Z
1C14 18F2       JR    1C0D


10 CLS:PRINT"un instant !"
20 FOR I=&H1C00 TO &H1C15
30 READ A$:POKE I,VAL("&H"+A$)
40 NEXT
50 DATA 06,FF,21,00,80,7E,A8,77,7C,FE,97
,28,03,23,18,F5,7D,FE,FF,C8,18,F2
60 SCREEN 5
70 PRINT"INVERSE VIDEO"
80 FOR I=0 TO 100
90 PSET (I,50):NEXT
100 EXEC&H1C00
110 FOR I=0 TO 500:NEXT:GOTO 100
```

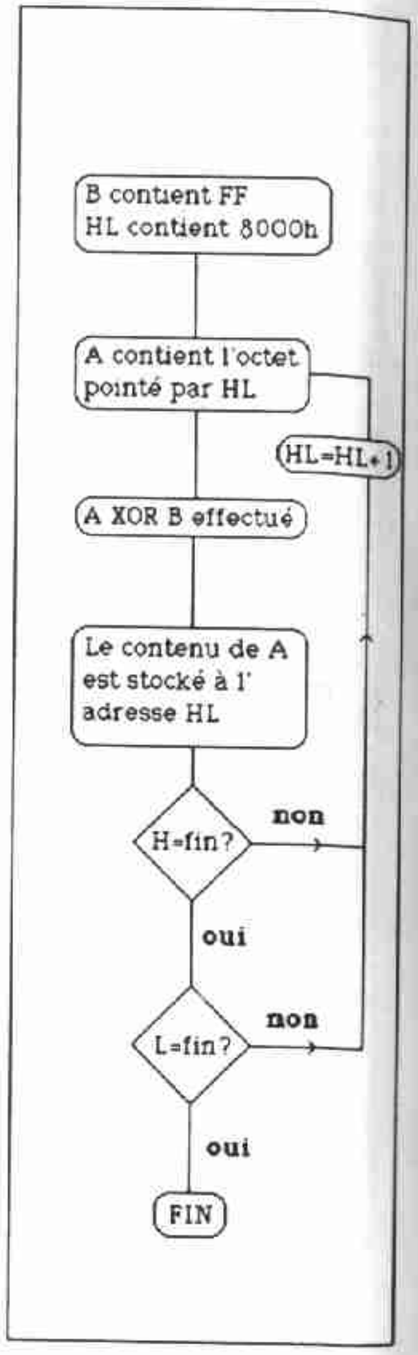


FIGURE 32 : INVERSION VIDEO

INVERSION DE L'AFFICHEUR LCD

On reprend **le même principe** pour inverser l'afficheur LCD que pour inverser l'afficheur VIDEO .

On inverse **octet par octet le contenu de la RAM LCD** à l'aide de la fonction XOR mais un détail diffère ... En effet , vous vous rappelez sans doute que la mémoire d'écran LCD est adressée non pas par le NSC 800 mais par notre cher **sous-processeur T6834** !! Résoudre ce problème nous amène à utiliser **des routines d'écriture et de lecture dans la RAM LCD** décrites dans le chapitre 9 .

Par contre , afin de rendre le programme **plus véloce** , nous n'avons pas réellement tenu compte de l'adressage de la RAM ... Quoi ??? Ne vous affolez surtout pas !!

En effet , nous inversons tous les octets compris entre les adresses **E000h et E1FEh** . Or , nous pouvons très bien nous passer d'inverser les octets **E00Fh , E01Fh et E02Fh** ... Mais , étant donné que les tests sur ces valeurs prendraient beaucoup plus de temps que l'inversion pure et simple de ces trois octets , nous avons préféré privilégier la vitesse d'exécution . Peu de personnes dénigreraient cette astuce connaissant la lenteur extrême de l'afficheur LCD .

ATTENTION ! Sur certaines versions du CANON X-07 , il arrive que des octets échappent à l'inversion si le T6834 se trouve débordé par le flux d'informations transitant vers ses circuits ... Pour pallier à ce problème , les utilisateurs possédant un X-07 fainéant peuvent se servir de la fonction N°13h décrite au paragraphe 7.6.1 .

Longueur de la routine : **29 octets** .
Implantation de la routine : **de 1C00h à 1C1Ch** .

```
10 '[
15 '*INVERSE LCD
20 'ORG $1C00
30 'LD HL,$DFFF:*RAM LCD -1
31 '#1E INC HL
40 'CALL $E348:*LECTURE RAM LCD
42 'LD B,$FF
50 'XOR B
60 'CALL $E334:*ECRITURE RAM LCD
65 'PUSH HL
70 'LD A,H
80 'CP $E1:*OCTET FORT AD.RAM LCD
90 'JR Z,#FI
95 'POP HL
110 'JR #1E
120 '#FI LD A,L
130 'CP $FE:*OCTET FAIBLE AD.RAM LCD
135 'POP HL
140 'RET Z
160 'JR #1E
170 ']
```

```
1C00 21FFDF      LD    HL,DFFF
1C03 23          INC   HL
1C04 CD48E3      CALL  E348
1C07 06FF        LD    B,FF
1C09 A8          XOR   B
1C0A CD34E3      CALL  E334
1C0D E5          PUSH  HL
1C0E 7C          LD    A,H
1C0F FEE1        CP    E1
1C11 2803        JR    Z,1C16
1C13 E1          POP   HL
1C14 18ED        JR    1C03
1C16 7D          LD    A,L
1C17 FEFE        CP    FE
1C19 E1          POP   HL
1C1A C8          RET   Z
1C1B 18E6        JR    1C03
```

```
10 CLS :PRINT"un instant"
20 FOR I=&H1C00 TO &H1C1B
30 READ A$:POKEI,VAL("&H"+A$)
40 NEXT
45 EXEC&H1C00
50 DATA 21,FF,DF,23,CD,48,E3,06,FF,A8,CD
,34,E3,E5,7C,FE,E1,28,03,E1,18,ED,7D
60 DATA FE,FE,E1,C8,18,E6
```

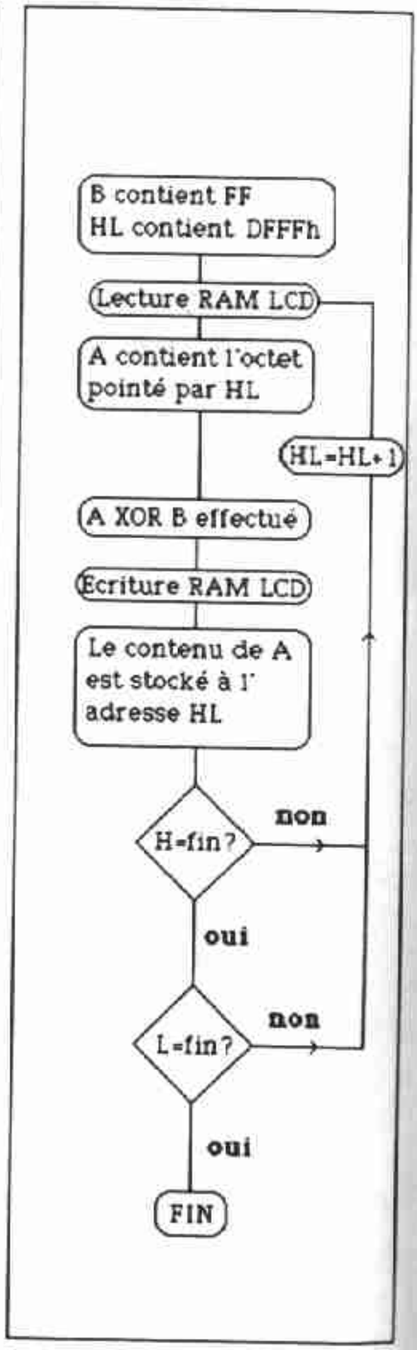


FIGURE 33 : INVERSION LCD

AFFICHAGE D'UN TITRE

Ce programme bloque la première ligne de l'écran en y affichant un **message personnel** . Ce **"titre"** ne peut être effacé par un scrolling (déroulement , en français ...) de l'affichage mais seulement par un **CLS** .

Le principe est des plus classiques . En effet , après avoir effacé l'écran par l'ordre **CE9Eh** (Voir chapitre 9) , on écrit dans la MEV d'unité centrale secondaire la ligne de début du SCROLL (à l'adresse **C09Eh**) et la ligne de fin de ce SCROLL (à l'adresse **C096h**) .

Ensuite , on peut afficher le message par la routine **FEF7h** étudiée au paragraphe 9.6 . Le programme exposé affiche le message "*** PROGRAMME ***" mais rien ne vous empêche d'en choisir un autre . N'oubliez surtout pas de terminer votre message par le code "00" pour que la routine FEF7h puisse en déterminer la fin .

Longueur de la routine : **44 octets** .
Implantation de la routine : **de 1C00h à 1C2Bh** .

```
10 'I
15 '*TITRE
20 'ORG $:C00
30 'CALL $CE9E :*EFFACE L'ECRAN
40 'LD A,$01 :*DEBUT DEFILEMENT
50 'LD HL,$C09E:*AD.EN RAM
60 'CALL $E334
70 'LD A,$03 :*FIN DE DEFILEMENT
80 'LD HL,$C096:*AD.EN RAM
90 'CALL $E334
100 'LD HL,#ME
110 'CALL $FEF7:*AFFICHAGE DU TITRE
120 'RET
130 '#ME DEFM  *** PROGRAMME ***
135 'DEFB $00
140 'J
```

```
1C00 CD9ECE       CALL  CE9E
1C03 3E01         LD    A,01
1C05 219EC0       LD    HL,C09E
1C08 CD34E3       CALL  E334
1C0B 3E03         LD    A,03
1C0D 2196C0       LD    HL,C096
1C10 CD34E3       CALL  E334
1C13 211A1C       LD    HL,1C1A
1C16 CDF7FE       CALL  FEF7
1C19 C9           RET
```

```
1C1A:20 2A 2A 2A 20 50 52 4F    *** PRO
1C22:47 52 41 4D 4D 45 20 2A   GRAMME *
1C2A:2A 2A 00 00 00 00 00 00   **......
```

```
10 CLS:PRINT"un instant !"
20 FOR I=&H1C00 TO &H1C2C
30 READ A$:POKE I,VAL("&H"+A$)
40 NEXT I:EXEC&H1C00
50 DATA CD,9E,CE,3E,01,21,9E,C0,CD,34,E3
,3E,03,21,96,C0,CD,34,E3,21,1A,1C
60 DATA CD,F7,FE,C9,20,2A,2A,2A,20,50,52
,4F,47,52,41,4D,4D,45,20,2A,2A,2A,00
```

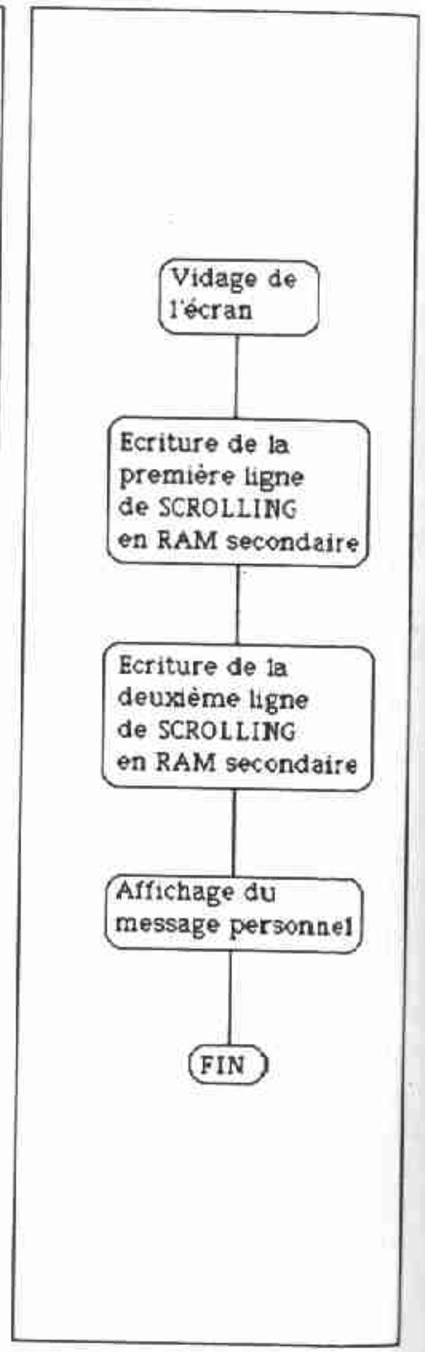


FIGURE 34 : AFFICHAGE d'un TITRE

COPYRIGHT PERSONNEL

Cette routine va vous autoriser à afficher **un message personnel dès la mise en route du X-07** ... L'imagination est au pouvoir !!

Le principe de la routine est basé sur la structure logicielle de démarrage du CANON . Effectivement , on utilise **le mot-clé "love"** (Voir le paragraphe 7.4) pour envoyer le pointeur de la machine vers un sous-programme personnel d'affichage .

Dans le cas présent , on a uniquement utilisé l'appel à une routine ... Si on avait voulu obtenir la même chose en démarrant après un SLEEP , il aurait fallu se servir de la troisième adresse positionnée après le mot-clé "love" .

La routine est entièrement relogeable ... ATTENTION ! Le mot-clé "love" n'est testé que tous les **800h octets** (donc tous les 2048 octets) à partir de **l'adresse 2000h** . On en conclut que les adresses possibles se résument à : **2000h , 2800h , 3000h , 3800h , 4000h , 4800h , 5000h** .

De plus , il est primordial de modifier les adresses des sous-programmes en fonction de l'emplacement du programme personnel .

Par exemple , si la routine avait débuté en **3000h** , elle se serait présentée de la manière suivante :

```
ORG $3000
DEFM love
DEFB $0A,30
DEFB $19,30
DEFB $19,30
```

Référez-vous vite à la figure 35 ... Votre CANON va dorénavant posséder un visage différent !!

Longueur de la routine : **29 octets** .
Implantation de la routine : **de 200Ah à 2026h** .

```
200A CD9ECE       CALL   CE9E
200D 210203       LD     HL,0302
2010 22B800       LD     (00BB),HL
2013 211C20       LD     HL,201C
2016 CDF7FE       CALL   FEF7
2019 C33DF2       JP     F23D

2000:6C 6F 76 65 0A 20 19 20    love. .
2008:19 20 CD 9E CE 21 02 03    . M.N..

201A:3D F2 76 69 76 65 20 6C    =rvive l
2022:65 20 58 30 37 00 00 00    e X07...
```

```
10 '[
20 '*COPYRIGHT
30 'ORG $2000
40 'DEFM love
50 'DEFB $0A,20:*1ER SOUS PROG.
60 'DEFB $19,20:*2EME SOUS PROG.
70 'DEFB $19,20:*SOUS PROG DE SLEEP
80 'CALL $CE9E:*EFFACE L'ECRAN
90 'LD HL,*0302
100 'LD [$00B8],HL:*POSITION DE CURSEUR
110 'LD HL,#ME
120 'CALL $FEF7
130 'JP F23D :*ATTENTE CURSEUR
140 '#ME DEFM vive le X07
150 'DEFB $00
160 ']
```

```
10 CLS:PRINT"un instant"
20 FOR I=&H2000 TO &H2027
30 READ A$:POKE I,VAL("&H"+A$)
40 NEXT I
50 OFF
60 DATA 6C,6F,76,65,0A,20,19,20,19,20,CD
,9E,CE,21,02,03,22,B8,00,21,1C,20
70 DATA CD,F7,FE,C3,3D,F2,76,69,76,65,20
,6C,65,20,58,30,37,00,CC,CC,CC
```

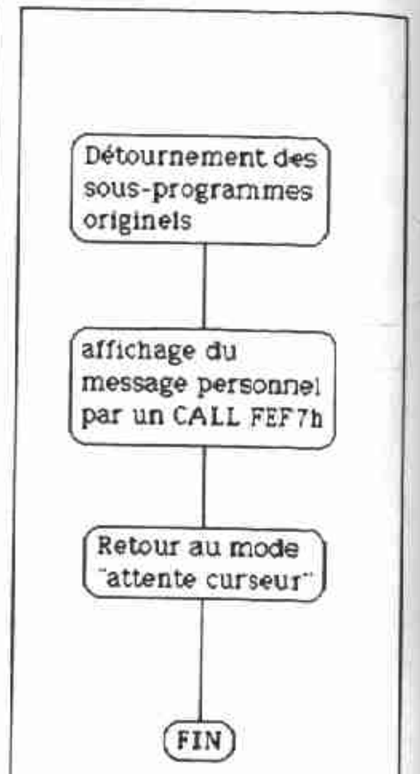


FIGURE 35 : COPYRIGHT

ECRITURE SUR LA X-710

Ce programme a pour but **d'écrire un message quelconque sur l'imprimante X-710** . Il est très simple d'emploi et son intérêt réside dans la façon d'utiliser les routines concernant la X-710 .

Le principe est donc relativement simple . L'imprimante est initialisée par la routine **CFB7h** . Ensuite , le message est écrit sur l'imprimante avec la routine **CED6h** .

On peut noter que dès la découverte d'un code "00" par l'interpréteur , le message est terminé et la X-710 arrête d'écrire .

Longueur de la routine : **24 octets** .
Implantation de la routine : **de 1C00h à 1C17h** .

```
10 '[
20 'ORG $1C00
30 '*ECRITURE
40 'CALL $CFB7:*INITIAL.DE L'IMP.
50 'LD HL,#1B
60 '#1E LD A,(HL)
70 'CP $00 :*FIN DE CHAINE
80 'RET Z
90 'PUSH AF
100 'PUSH BC
110 'PUSH DE
120 'PUSH HL
130 'CALL $CED6:*ECRITURE SUR IMP.
140 'POP HL
150 'POP DE
160 'POP BC
170 'POP AF
180 'INC HL
190 'JR #1E
200 '#1B DEFM BONJOUR DE X07
210 'DEFB $00
220 ']

1C00 CDB7CF      CALL  CFB7
1C03 211B1C      LD    HL,1C1B
1C06 7E          LD    A,(HL)
1C07 FE00        CP    00
1C09 C8          RET   Z
1C0A F5          PUSH  AF
1C0B C5          PUSH  BC
1C0C D5          PUSH  DE
1C0D E5          PUSH  HL
1C0E CDD6CE      CALL  CED6
1C11 E1          POP   HL
1C12 D1          POP   DE
1C13 C1          POP   BC
1C14 F1          POP   AF
1C15 23          INC   HL
1C16 18EE        JR    1C06
```

```
10 CLS:PRINT"un instant !"
20 FOR I=&H1C00 TO &H1C26
30 READ A$:POKE I,VAL("&H"+A$)
40 NEXT I
60 DATA CD,B7,CF,21,1B,1C,7E,FE,00,C8,F5
,C5,D5,E5,CD,D6,CE,E1,D1,C1,F1,23
70 DATA 18,EE,42,4F,4E,4A,4F,55,52,20,44
,45,20,58,30,37,00
```

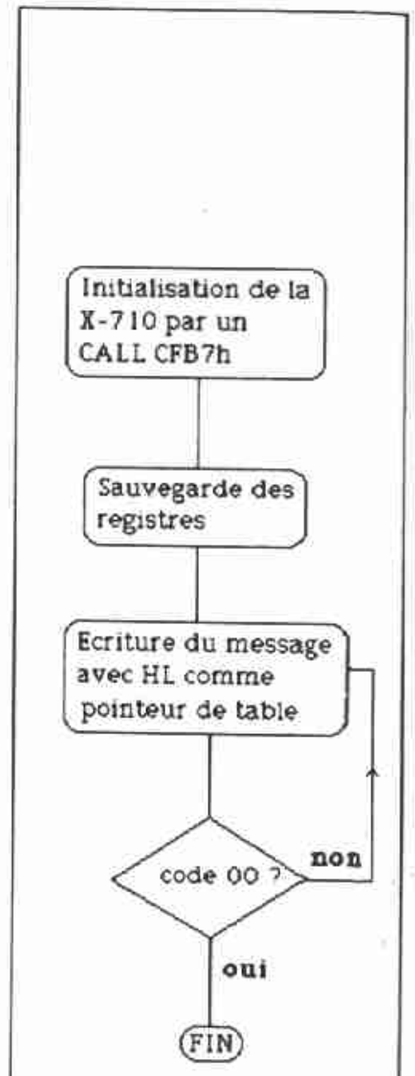


FIGURE 36 : ECRITURE sur X-710

BRUITAGES DIVERS

Ce programme va surement enthousiasmer plus d'un programmeur ... En effet , en utilisant **les ports de sortie du NSC 800** , nous allons déclencher le BUZZER d'une façon tout à fait particulière . Que les adhérents du CLUB C7 se rappellent de la gazette N°1 , d'un certain article qui émettait **des sons assez inattendus** !!

Le principe est le suivant ... Nous allons utiliser **trois ports** :

_ **Le port F4** qui met en route le BUZZER . On y place la configuration de bits suivante : le bit BZ ON est mis à1 , les bits MD1 et MD0 sont forcés à 1 (sortie sur le BUZZER) et le bit BRGST est aussi positionné à 1 (le générateur de bauds peut compter ...) .

Donc , le port F4 contient la valeur **4Eh** .

_ **Les ports F2 et F3** servent à placer la vitesse d'émission des impulsions du BUZZER dans le générateur de bauds . Cette vitesse est codée sur 12 bits comme nous l'avons déjà fait remarquer : 8 octets dans le port F2 et 4 autres dans le port F3 .

En modifiant les valeurs contenues dans ces différents ports , le son varie indéfiniment !!

Une **routine TEMPO** est incluse dans le programme . Elle a pour but de ralentir le processeur pour que les sons soient audibles .

Comment faire pour arrêter le BUZZER ? Tout simplement placer **le code "00" dans le port F4** .

Longueur de la routine : **44 octets** .
Implantation de la routine : **de 1C00h à 1C2Bh** .

```
10 '[
20 'ORG $1C00
30 '*BRUIT
35 'LD A,$00
40 'LD B,$0F
50 'OUT ($F3),A
60 'LD A,$4E
70 'OUT ($F4),A
80 '#2E LD A,$FF
90 '#1E OUT ($F2),A
100 'DEC A
110 'JR Z,#F3
120 'CALL #TE
130 'JR #1E
140 '#F3 LD A,B
150 'DEC A
160 'JR Z,#FI
170 'OUT ($F3),A
180 'LD B,A
190 'JR #2E
195 '*---------------
196 '*      ARRET
197 '*---------------
200 '#FI LD A,$00
210 'OUT ($F4),A
220 'RET
225 '*---------------
226 '*     TEMPO
227 '*---------------
230 '#TE PUSH AF
240 'LD A,$80
250 '#LO DEC A
260 'JR NZ,#LO
265 'POP AF
270 'RET
280 ']

10 PRINT"un instant ?"
20 FOR I=&H1C00 TO &H1C2B
30 READ A$:POKE I,VAL("&H"+A$)
40 NEXT
50 EXEC&H1C00
60 DATA 3E,00,06,0F,D3,F3,3E,4E,D3,F4,3E
,FF,D3,F2,3D,28,05,CD,24,1C
70 DATA 18,F6,78,3D,28,05,D3,F3,47,18,EB
,3E,00,D3,F4,C9,F5,3E,00,3D,20,FD,F1,C9
```

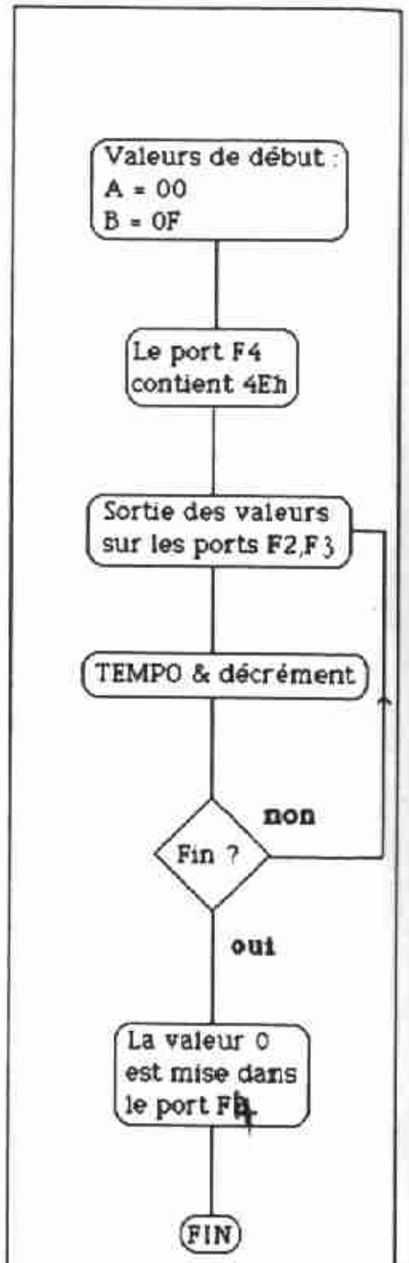


FIGURE 37 : BRUITAGES

REDEFINITION DE TOUCHES

Le but de ce dernier programme est **d'assigner à une touche quelconque du clavier un message** . Vous percevez surement l'utilité de cette routine : **le clavier entier peut être redéfini en AZERTY , par exemple** .

Ici , **la touche ~ (Tildé espagnol) provoquera l'affichage de "X-07"** à chaque appui .

Le principe adopté est d'appeler **la routine d'affichage** à chaque appui d'une touche . Cette routine est appelée par une adresse se trouvant en zone système , exactement en **9Fh** (Voir le paragraphe 8.3) .

En modifiant l'adresse de saut , on fait passer cette routine par un petit programme personnel comparant la touche appuyée avec le code ASCII se trouvant dans le registre A (Code ASCII de la touche redéfinie) . Si les deux codes sont égaux , on exécute l'affichage du message défini , sinon on retourne à la routine initiale (**CEB1h**) qui affichera le caractère contenu dans l'accumulateur .

Le programme se présente donc en **deux parties** :

- La première partie modifie l'adresse de saut se trouvant à l'adresse 9Fh.
- La deuxième partie effectue la comparaison et affiche le message si cela se révèle nécessaire .

Si vous regardez l'organigramme , vous remarquerez qu'il n'y a pas de FIN à ce programme ... Pourquoi ? Tout simplement parce que la routine est **interactive** : elle décide d'elle-même si elle doit fonctionner ou pas ...

Longueur de la routine : **32 octets** .
Implantation de la routine : **de 1C00h à 1C1Fh** .

```
10 '[
20 'ORG $1C00
30 '*REDEF DE TOUCHE
40 '*CHANGEMENT DE HOOCK
50 'LD HL,#DB
60 'LD ($A0),HL
70 'RET
80 '*---------------
90 '*   PROGRAMME
100 '*---------------
110 '#DB PUSH AF
120 'CP "~"
130 'JR Z,#AF
140 'POP AF
150 'JP $C1BE
160 '#AF PUSH HL
170 'LD HL,#ME
180 'CALL $D580
190 'POP HL
200 'POP AF
210 'JP $F23D
220 '#ME DEFM  X07
230 'DEFB $00
240 ']
```

```
1C00 21071C      LD    HL,1C07
1C03 22A000      LD    (00A0),HL
1C06 C9          RET
1C07 F5          PUSH  AF
1C08 FE7E        CP    7E
1C0A 2804        JR    Z,1C10
1C0C F1          POP   AF
1C0D C3BEC1      JP    C1BE
1C10 E5          PUSH  HL
1C11 211C1C      LD    HL,1C1C
1C14 CD80D5      CALL  D580
1C17 E1          POP   HL
1C18 F1          POP   AF
1C19 C33DF2      JP    F23D
```

```
10 PRINT"un instant"
20 FOR I=&H1C00 TO &H1C20
30 READ A$:POKEI,VAL("&H"+A$)
40 NEXT
50 EXEC&H1C00:PRINT"APPUYEZ SUR ~"
60 DATA 21,07,1C,22,A0,00,C9,F5,FE,7E,28
,04,F1,C3,BE,C1,E5,21,1C,1C,CD,80,D5
70 DATA E1,F1,C3,3D,F2,20,58,30,37,00
```

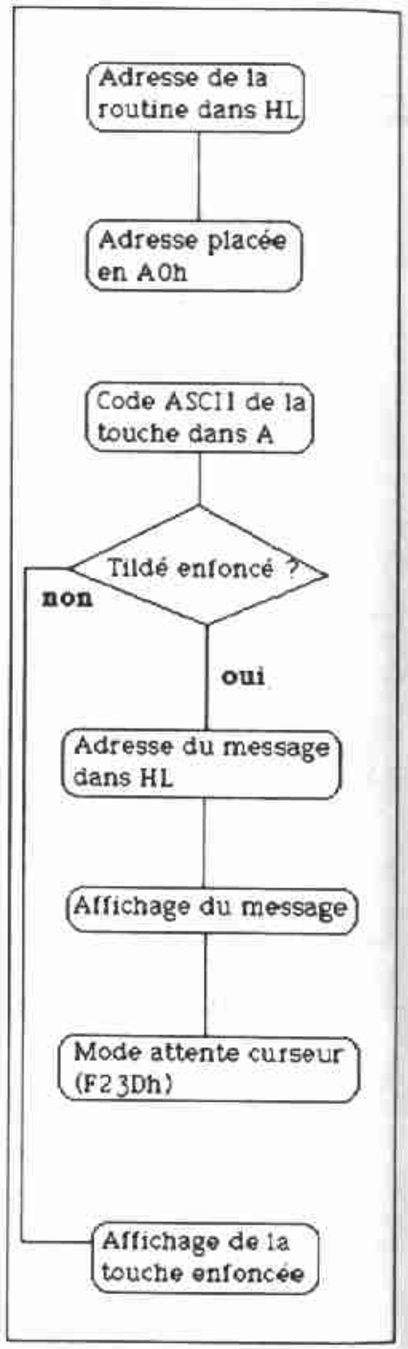


FIGURE 38 : REDEFINITION

CONCLUSION

OCTOBRE 1985 , QUELQUE PART EN FRANCE , 8 h 10 ...

CANDORE s'est assoupi ... Sur la table , plusieurs feuilles gribouillées de sa main trainent . Un micro portatif **CANON X-07** trône près de lui agité par des spasmes bizarres . En effet , l'écran clignote intensément avec une vélocité phénoménale ; le BUZZER émet des sons totalement inconnus du BASIC classique ; l'imprimante trace des lignes à une vitesse effarante !!

Soudain , le téléviseur , un peu en retrait , projette ses couleurs sur les multiples facettes de la pièce , doublées d'un message triomphant : **EUREKA** !!

OCTOBRE 1985 , QUELQUE PART EN FRANCE , 16 h 45 ...

Quelques heures plus tard , CANDORE se réveille et retrouve joyeux ce petit monde qu'il vient d'animer . Il a réussi à réveiller toutes les possibilités sommeillant au fin fond de son matériel informatique .

Tout à coup , un flash !! CANDORE se rend compte que son rêve s'est réalisé ! Par l'intermédiaire de ce livre magique , le vieux sage ASSEMBLUS lui a révélé tous les secrets de son X-07 ...

Tout en jubilant intérieurement , CANDORE pense à la chance qu'il a d'avoir pu lire cet ouvrage : "le CANON X-07 et l'ASSEMBLEUR" .

Qui sait ... Peut-être qu'un jour une suite paraîtra car , CANDORE le pressent , le X-07 n'a pas fini de nous éblouir !

A. TONIC

ANNEXES

ANNEXE 1

LISTE DES CODES

CODE	MNEMONIQUE	INDICATEURS
8E	ADC A,(HL)	
DD8Edpl	ADC A,(IX + dpl	
FD8Edpl	ADC A,(IY + dpl)	
8F	ADC A,A	
88	ADC A,B	
89	ADC A,C	
8A	ADC A,D	
8B	ADC A,E	
8C	ADC A,H	
8D	ADC A,L	
CEv	ADC A,v	
ED4A	ADC HL,BC	
ED5A	ADC HL,DE	
ED6A	ADC HL,HL	
ED7A	ADC HL,SP	
86	ADD A,(HL)	
DD86dpl	ADD A,(IX + dpl)	
FD86dpl	ADD A,(IY + dpl)	
87	ADD A,A	modifiés
80	ADD A,B	
81	ADD A,C	
82	ADD A,D	
83	ADD A,E	
84	ADD A,H	
85	ADD A,L	
C6v	ADD A,v	
09	ADD HL,BC	
19	ADD HL,DE	
29	ADD HL,HL	
39	ADD HL,SP	
DD09	ADD IX,BC	
DD19	ADD IX,DE	
DD29	ADD IX,IX	
DD39	ADD IX,SP	
FD09	ADD IY,BC	
FD19	ADD IY,DE	
FD29	ADD IY,IY	
FD39	ADD IY,SP	

A6	AND (HL)	
DDA6dpl	AND (IX + dpl)	
FDA6dpl	AND (IY+ dpl)	C=N=0
A7	AND A	P=parité
A0	AND B	H=1
A1	AND C	S et Z modifiés
A2	AND D	
A3	AND E	
A4	AND H	
A5	AND L	
E6v	AND v	

CB46	BIT 0,(HL)	
DDCBdpl46	BIT 0,(IX + dpl)	
FDCBdpl46	BIT 0,(IY + dpl)	
CB47	BIT 0,A	
CB40	BIT 0,B	
CB41	BIT 0,C	
CB42	BIT 0,D	
CB43	BIT 0,E	
CB44	BIT 0,H	
CB45	BIT 0,L	
CB4E	BIT 1,(HL)	
DDCBdpl4E	BIT 1,(IX + dpl)	
FDCBdpl4E	BIT 1,(IY + dpl)	
CB4F	BIT 1,A	
CB48	BIT 1,B	
CB49	BIT 1,C	Z modifié
CB4A	BIT 1,D	N=0 , H=1
CB4B	BIT 1,E	C inchangé
CB4C	BIT 1,H	
CB4D	BIT 1,L	
CB56	BIT 2,(HL)	
DDCBdpl56	BIT 2,(IX + dpl)	
FDCBdpl56	BIT 2,(IY + dpl)	
CB57	BIT 2,A	
CB50	BIT 2,B	
CB51	BIT 2,C	
CB52	BIT 2,D	
CB53	BIT 2,E	
CB54	BIT 2,H	
CB55	BIT 2,L	
CB5E	BIT 3,(HL)	
DDCBdpl5E	BIT 3,(IX + dpl)	
FDCBdpl5E	BIT 3,(IY + dpl)	
CB5F	BIT 3,A	

CB58	BIT 3,B	
CB59	BIT 3,C	
CB5A	BIT 3,D	
CB5B	BIT 3,E	
CB5C	BIT 3,H	
CB5D	BIT 3,L	
CB66	BIT 4,(HL)	
DDCBdpl66	BIT 4,(IX + dpl)	
FDCBdpl66	BIT 4,(IY + dpl)	
CB67	BIT 4,A	
CB60	BIT 4,B	
CB61	BIT 4,C	
CB62	BIT 4,D	
CB63	BIT 4,E	
CB64	BIT 4,H	
CB65	BIT 4,L	
CB6E	BIT 5,(HL)	
DDCBdpl6E	BIT 5,(IX + dpl)	
FDCBdpl6E	BIT 5,(IY + dpl)	
CB6F	BIT 5,A	Z modifié
CB68	BIT 5,B	N=0 , H=1
CB69	BIT 5,C	C inchangé
CB6A	BIT 5,D	
CB6B	BIT 5,E	
CB6C	BIT 5,H	
CB6D	BIT 5,L	
CB76	BIT 6,(HL)	
DDCBdpl76	BIT 6,(IX + dpl)	
FDCBdpl76	BIT 6,(IY + dpl)	
CB77	BIT 6,A	
CB70	BIT 6,B	
CB71	BIT 6,C	
CB72	BIT 6,D	
CB73	BIT 6,E	
CB74	BIT 6,H	
CB75	BIT 6,L	
CB7E	BIT 7,(HL)	
DDCBdpl7E	BIT 7,(IX + dpl)	
FDCBdpl7E	BIT 7,(IY + dpl)	
CB7F	BIT 7,A	
CB78	BIT 7,B	
CB79	BIT 7,C	
CB7A	BIT 7,D	
CB7B	BIT 7,E	
CB7C	BIT 7,H	
CB7D	BIT 7,L	

DCnn	CALL C,nn	
FCnn	CALL M,nn	
D4nn	CALL NC,nn	
CDnn	CALL nn	
C4nn	CALL NZ,nn	non-affectés
F4nn	CALL P,nn	
ECnn	CALL PE,nn	
E4nn	CALL PO,nn	
CCnn	CALL Z,nn	
3F	CCF	C modifié , N=0
BE	CP (HL)	
DDBEdpl	CP (IX + dpl)	
FDBEdpl	CP (IY + dpl)	
BF	CP A	
B8	CP B	
B9	CP C	N=1 , les autres
BA	CP D	sont modifiés
BB	CP E	
BC	CP H	
BD	CP L	
FEv	CP v	
EDA9	CPD	N=1 , C inchangé P/V=1 sauf si BC =0, Z=1 si A=(BC)
EDB9	CPDR	C inchangé , N=1 S et H modifiés V=0 si BC=0 Z=1 si égalité
EDA1	CPI	idem que CPD
EDB1	CPIR	idem que CPDR
2F	CPL	N=H=1, les autres sont inchangés

27	DAA	N inchangé , les autres modifiés
35	DEC (HL)	
DD35dpl	DEC (IX + dpl)	
FD35dpl	DEC (IY + dpl)	
3D	DEC A	
05	DEC B	
0B	DEC BC	
0D	DEC C	Pour les DEC rr , non-modifiés . Sinon, C inchangé autres modifiés .
15	DEC D	
1B	DEC DE	
1D	DEC E	
25	DEC H	
2B	DEC HL	
DD2B	DEC IX	
FD2B	DEC IY	
2D	DEC L	
3B	DEC SP	
F3	DI	non-affectés
10dpl	DJNZ label	non-affectés
FB	EI	non-affectés
E3	EX (SP),HL	non-affectés
DDE3	EX (SP),IX	
FDE3	EX (SP),IY	
08	EX AF,AF'	état du reg. F' non-affectés
EB	EX DE,HL	
D9	EXX	
76	HALT	non-modifiés
ED46	IM 0	non-affectés
ED56	IM 1	
ED5E	IM 2	

ED78	IN A,(C)	
DBv	IN A,(v)	Non-affectés pour IN A,(v) . Sinon , S,Z,H et P modifiés , N=0 , C inchangé .
ED40	IN B,(C)	
ED48	IN C,(C)	
ED50	IN D,(C)	
ED58	IN E,(C)	
ED60	IN H,(C)	
ED68	IN L,(C)	
34	INC (HL)	
DD34dpl	INC (IX + dpl)	
FD34dpl	INC (IY + dpl)	
3C	INC A	
04	INC B	
03	INC BC	
0C	INC C	
14	INC D	
13	INC DE	Idem que DEC
1C	INC E	
24	INC H	
23	INC HL	
DD23	INC IX	
FD23	INC IY	
2C	INC L	
33	INC SP	
EDAA	IND	N=1 , C inchangé Z=1 si B=0 , les autres sont indéterminés .
EDB.	INDR	C inchangé , N et Z=1 , les autres sont quelconques
EDA2	INI	idem que IND
EDB2	INIR	idem que INDR
E9	JP (HL)	non-affectés
DDE9	JP (IX)	

FDE9	JP (IY)	
DAnn	JP C,nn	
FAnn	JP M,nn	
D2nn	JP NC,nn	
C3nn	JP nn	
C2nn	JP NZ,nn	
F2nn	JP P,nn	
EAnn	JP PE,nn	non-affectés
E2nn	JP PO,nn	
CAnn	JP Z,nn	
38dpl	JR C,label	
18dpl	JR label	
30dpl	JR NC,label	
20dpl	JR NZ,label	
28dpl	JR Z,label	
02	LD (BC),A	
12	LD (DE),A	
77	LD (HL),A	
70	LD (HL),B	
71	LD (HL),C	
72	LD (HL),D	
73	LD (HL),E	
74	LD (HL),H	
75	LD (HL),L	
36v	LD (HL),v	
DD77dpl	LD (IX + dpl),A	
DD70dpl	LD (IX + dpl),B	
DD71dpl	LD (IX + dpl),C	non-affectés
DD72dpl	LD (IX + dpl),D	
DD73dpl	LD (IX + dpl),E	
DD74dpl	LD (IX + dpl),H	
DD75dpl	LD (IX + dpl),L	
DD36dplv	LD (IX + dpl),v	
FD77dpl	LD (IY + dpl),A	
FD70dpl	LD (IY + dpl),B	
FD71dpl	LD (IY + dpl),C	
FD72dpl	LD (IY + dpl),D	
FD73dpl	LD (IY + dpl),E	
FD74dpl	LD (IY + dpl),H	
FD75dpl	LD (IY + dpl),L	
FD36dplv	LD (IY + dpl),v	
32nn	LD (nn),A	
ED43nn	LD (nn),BC	
ED53nn	LD (nn),DE	
22nn	LD (nn),HL	

DD22nn	LD (nn),IX	
FD22nn	LD (nn),IY	
ED73nn	LD (nn),SP	
0A	LD A,(BC)	
1A	LD A,(DE)	
7E	LD A,(HL)	
DD7Edpl	LD A,(IX + dpl)	
FD7Edpl	LD A,(IY + dpl)	
3Ann	LD A,(nn)	
7F	LD A,A	
78	LD A,B	
79	LD A,C	
7A	LD A,D	
7B	LD A ,E	
7C	LD A,H	
ED57	LD A,I	
7D	LD A,L	
3Ev	LD A,v	
ED5F	LD A,R	
46	LD B,(HL)	non-affectés
DD46dpl	LD B,(IX + dpl)	
FD46dpl	LD B,(IY + dpl)	
47	LD B,A	
40	LD B,B	
41	LD B,C	
42	LD B,D	
43	LD B,E	
44	LD B,H	
45	LD B,L	
06v	LD B,v	
ED4Bnn	LD BC,(nn)	
01vv	LD BC,vv	
4E	LD C,(HL)	
DD4Edpl	LD C,(IX + dpl)	
FD4Edpl	LD C,(IY + dpl)	
4F	LD C,A	
48	LD C,B	
49	LD C,C	
4A	LD C,D	
4B	LD C,E	
4C	LD C,H	
4D	LD C,L	
0Ev	LD C,v	
56	LD D,(HL)	
DD56dpl	LD D,(IX + dpl)	
FD56dpl	LD D,(IY + dpl)	

57	LD D,A	
50	LD D,B	
51	LD D,C	
52	LD D,D	
53	LD D,E	
54	LD D,H	
55	LD D,L	
16v	LD D,v	
ED5Bnn	LD DE,(nn)	
11vv	LD DE,vv	
5E	LD E,(HL)	
DD5Edpl	LD E,(IX + dpl)	
FD5Edpl	LD E,(IY + dpl)	
5F	LD E,A	
58	LD E,B	
59	LD E,C	
5A	LD E,D	
5B	LD E,E	
5C	LD E,H	
5D	LD E,L	
1Ev	LD E,v	
66	LD H,(HL)	non-affectés
DD66dpl	LD H,(IX + dpl)	
FD66dpl	LD H,(IY + dpl)	
67	LD H,A	
60	LD H,B	
61	LD H,C	
62	LD H,D	
63	LD H,E	
64	LD H,H	
65	LD H,L	
26v	LD H,v	
2Ann	LD HL,(nn)	
21vv	LD HL,vv	
ED47	LD I,A	
DD2Ann	LD IX,(nn)	
DD21vv	LD IX,vv	
FD2Ann	LD IY,(nn)	
FD21vv	LD IY,vv	
6E	LD L,(HL)	
DD6Edpl	LD L,(IX + dpl)	
FD6Edpl	LD L,(Y + dpl)	
6F	LD L,A	
68	LD L,B	
69	LD L,C	
6A	LD L,D	

6B	LD L,E	
6C	LD L,H	
6D	LD L,L	
2Ev	LD L,v	
ED47	LD R,A	
ED7Bnn	LD SP,(nn)	non-affectés
F9	LD SP,HL	
DDF9	LD SP,IX	
FDF9	LD SP,IY	
31vv	LD SP,vv	
EDA8	LDD	S,Z,C inchangés N=H=0 P/V=0 si BC=0 sinon P/V=1
EDB0	LDIR	N=P=H=0 autres inchangés
ED44	NEG	N=1 autres modifiés
00	NOP	non-affectés
B6	OR (HL)	
DDB6dpl	OR (IX + dpl)	
FDB6dpl	OR (IY + dpl)	
B7	OR A	
B0	OR B	
B1	OR C	idem que AND
B2	OR D	
B3	OR E	
B4	OR H	
B5	OR L	
F6v	OR v	
EDBB	OTDR	idem que INIR
EDB3	OTIR	
ED79	OUT (C),A	
ED41	OUT (C),B	

ED49	OUT (C),C	
ED51	OUT (C),D	
ED59	OUT (C),E	
ED61	OUT (C),H	non-affectés
ED69	OUT (C),L	
D3v	OUT (v),A	
EDAB	OUTD	idem que IND
EDA3	OUTI	
F1	POP AF	
C1	POP BC	
D1	POP DE	
E1	POP HL	
DDE1	POP IX	
FDE1	POP IY	non-affectés sauf
F5	PUSH AF	pour le POP AF
C5	PUSH BC	
D5	PUSH DE	
E5	PUSH HL	
DDE5	PUSH IX	
FDE5	PUSH IY	
CB86	RES 0,(HL)	
DDCBdpl86	RES 0,(IX + dpl)	
FDCBdpl86	RES 0,(IY + dpl)	
CB87	RES 0,A	
CB80	RES 0,B	
CB81	RES 0,C	
CB82	RES 0,D	
CB83	RES 0,E	
CB84	RES 0,H	
CB85	RES 0,L	
CB8E	RES 1,(HL)	
DDCBdpl8E	RES 1,(IX + dpl)	
FDCBdpl8E	RES 1,(IY + dpl)	
CB8F	RES 1,A	
CB88	RES 1,B	
CB89	RES 1,C	non-modifiés
CB8A	RES 1,D	
CB8B	RES 1,E	
CB8C	RES 1,H	
CB8D	RES 1,L	
CB96	RES 2,(HL)	
DDCBdpl96	RES 2,(IX + dpl)	
FDCBdpl96	RES 2,(IY + dpl)	

CB97	RES 2,A	
CB90	RES 2,B	
CB91	RES 2,C	
CB92	RES 2,D	
CB93	RES 2,E	
CB94	RES 2,H	
CB95	RES 2,L	
CB9E	RES 3,(HL)	
DDCBdp19E	RES 3,(IX + dpl)	
FDCBdp19E	RES 3,(IY + dpl)	
CB9F	RES 3,A	
CB98	RES 3,B	
CB99	RES 3,C	
CB9A	RES 3,D	
CB9B	RES 3,E	
CB9C	RES 3,H	
CB9D	RES 3,L	
CBA6	RES 4,(HL)	
DDCBdplA6	RES 4,(IX + dpl)	
FDCBdplA6	RES 4,(IY + dpl)	
CBA7	RES 4,A	
CBA0	RES 4,B	
CBA1	RES 4,C	
CBA2	RES 4,D	
CBA3	RES 4,E	
CBA4	RES 4,H	
CBA5	RES 4,L	
CBAE	RES 5,(HL)	
DDCBdplAE	RES 5,(IX + dpl)	
FDCBdplAE	RES 5,(IY + dpl)	
CBAF	RES 5,A	non-modifiés
CBA8	RES 5,B	
CBA9	RES 5,C	
CBAA	RES 5,D	
CBAB	RES 5,E	
CBAC	RES 5,H	
CBAD	RES 5,L	
CBB6	RES 6,(HL)	
DDCBdplB6	RES 6,(IX + dpl)	
FDCBdplB6	RES 6,(IY + dpl)	
CBB7	RES 6,A	
CBB0	RES 6,B	
CBB1	RES 6,C	
CBB2	RES 6,D	
CBB3	RES 6,E	
CBB4	RES 6,H	

CBB5	RES 6,L	
CBBE	RES 7,(HL)	
DDCBdplBE	RES 7,(IX + dpl)	
FDCBdplBE	RES 7,(IY + dpl)	
CBBF	RES 7,A	
CBB8	RES 7,B	non-modifiés
CBB9	RES 7,C	
CBBA	RES 7,D	
CBBB	RES 7,E	
CBBC	RES 7,H	
CBBD	RES 7,L	
C9	RET	
D8	RET C	
F8	RET M	
D0	RET NC	
C0	RET NZ	
F0	RET P	non-affectés
E8	RET PE	
E0	RET PO	
C8	RET Z	
ED4D	RETI	
ED45	RETN	
CB16	RL (HL)	
DDCBdpl16	RL (IX + dpl)	
FDCBdpl16	RL (IY + dpl)	
CB17	RL A	H=N=0
CB10	RL B	S ,Z ,C ,P modifiés
CB11	RL C	
CB12	RL D	
CB13	RL E	
CB14	RL H	
CB15	RL L	
17	RLA	H=N=0 , C modifié
		S , Z , P inchangés
CB06	RLC (HL)	
DDCBdpl06	RLC (IX + dpl)	
FDCBdpl06	RLC (IY + dpl)	
CB07	RLC A	
CB00	RLC B	N=H=0
CB01	RLC C	S ,Z ,C ,P modifiés
CB02	RLC D	
CB03	RLC E	

CB04	RLC H	
CB05	RLC L	
07	RLCA	
ED6F	RLD	N=H=0,C inchangé S , Z , P modifiés
CB1E	RR (HL)	
DDCBdpl1E	RR (IX + dpl)	
FDCBdpl1E	RR (IY + dpl)	
CB1F	RR A	
CB18	RR B	H=N=0
CB19	RR C	S ,Z ,C ,P modifiés
CB1A	RR D	
CB1B	RR E	
CB1C	RR H	
CB1D	RR L	
1F	RRA	idem que RLA
CB0E	RRC (HL)	
DDCBdp10E	RRC (IX + dpl)	
FDCBdp10E	RRC (IY + dpl)	
CB0F	RRC A	
CB08	RRC B	
CB09	RRC C	N=H=0
CB0A	RRC D	S ,Z ,C ,P modifiés
CB0B	RRC E	
CB0C	RRC H	
CB0D	RRC L	
0F	RRCA	
ED67	RRD	idem que RLD
C7	RST $00	
CF	RST $08	
D7	RST $10	
DF	RST $18	
E7	RST $20	non-affectés
EF	RST $28	
F7	RST $30	
FF	RST $38	
9E	SBC A,(HL)	
DD9Edpl	SBC A,(IX + dpl)	
FD9Edpl	SBC A,(IY + dpl)	
9F	SBC A,A	modifiés
98	SBC A,B	
99	SBC A,C	

9A	SBC A,D	
9B	SBC A,E	
9C	SBC A,H	
9D	SBC A,L	
DEv	SBC A,v	modifiés
ED42	SBC HL,BC	
ED52	SBC HL,DE	
ED62	SBC HL,HL	
ED72	SBC HL,SP	
37	SCF	C=1 , H=N=0
CBC6	SET 0,(HL)	
DDCBdpIC6	SET 0,(IX + dpl)	
FDCBdpIC6	SET 0,(IY + dpl)	
CBC7	SET 0,A	
CBC0	SET 0,B	
CBC1	SET 0,C	
CBC2	SET 0,D	
CBC3	SET 0,E	
CBC4	SET 0,H	
CBC5	SET 0,L	
CBCE	SET 1,(HL)	
DDCBdpICE	SET 1,(IX + dpl)	
FDCBdpICE	SET 1,(IY + dpl)	
CBCF	SET 1,A	
CBC8	SET 1,B	
CBC9	SET 1,C	inchangés
CBCA	SET 1,D	
CBCB	SET 1,E	
CBCC	SET 1,H	
CBCD	SET 1,L	
CBD6	SET 2,(HL)	
DDCBdpID6	SET 2,(IX + dpl)	
FDCBdpID6	SET 2,(IY + dpl)	
CBD7	SET 2,A	
CBD0	SET 2,B	
CBD1	SET 2,C	
CBD2	SET 2,D	
CBD3	SET 2,E	
CBD4	SET 2,H	
CBD5	SET 2,L	
CBDE	SET 3,(HL)	

DDCBdpIDE	SET 3,(IX + dpl)	
FDCBdpIDE	SET 3,(IY + dpl)	
CBDF	SET 3,A	
CBD8	SET 3,B	
CBD9	SET 3,C	
CBDA	SET 3,D	
CBDB	SET 3,E	
CBDC	SET 3,H	
CBDD	SET 3,L	
CBE6	SET 4,(HL)	
DDCBdpIE6	SET 4,(IX + dpl)	
FDCBdpIE6	SET 4,(IY + dpl)	
CBE7	SET 4,A	
CBE0	SET 4,B	
CBE1	SET 4,C	
CBE2	SET 4,D	
CBE3	SET 4,E	
CBE4	SET 4,H	
CBE5	SET 4,L	
CBEE	SET 5,(HL)	
DDCBdpIEE	SET 5,(IX + dpl)	
FDCBdpIEE	SET 5,(IY + dpl)	
CBEF	SET 5,A	inchangés
CBE8	SET 5,B	
CBE9	SET 5,C	
CBEA	SET 5,D	
CBEB	SET 5,E	
CBEC	SET 5,H	
CBED	SET 5,L	
CBF6	SET 6,(HL)	
DDCBdpIF6	SET 6,(IX + dpl)	
FDCBdpIF6	SET 6,(IY + dpl)	
CBF7	SET 6,A	
CBF0	SET 6,B	
CBF1	SET 6,C	
CBF2	SET 6,D	
CBF3	SET 6,E	
CBF4	SET 6,H	
CBF5	SET 6,L	
CBFE	SET 7,(HL)	
DDCBdpIFE	SET 7,(IX + dpl)	
FDCBdpIFE	SET 7,(IY + dpl)	
CBFF	SET 7,A	
CBF8	SET 7,B	
CBF9	SET 7,C	
CBFA	SET 7,D	

CBFB	SET 7,E	
CBFC	SET 7,H	
CBFD	SET 7,L	inchangés
CB26	SLA (HL)	
DDCBdpI26	SLA (IX + dpl)	
FDCBdpI26	SLA (IY + dpl)	
CB27	SLA A	
CB20	SLA B	
CB21	SLA C	H=N=0
CB22	SLA D	S ,Z ,C ,P modifiés
CB23	SLA E	
CB24	SLA H	
CB25	SLA L	
CB2E	SRA (HL)	
DDCBdpI2E	SRA (IX + dpl)	
FDCBdpI2E	SRA (IY + dpl)	
CB2F	SRA A	
CB28	SRA B	
CB29	SRA C	H=N=0
CB2A	SRA D	S ,Z ,C ,P modifiés
CB2B	SRA E	
CB2C	SRA H	
CB2D	SRA L	
CB3E	SRL (HL)	
DDCBdpI3E	SRL (IX + dpl)	
FDCBdpI3E	SRL (IY + dpl)	
CB3F	SRL A	
CB38	SRL B	
CB39	SRL C	H=N=0
CB3A	SRL D	S ,Z ,C ,P modifiés
CB3B	SRL E	
CB3C	SRL H	
CB3D	SRL L	
96	SUB (HL)	
DD96dpl	SUB (IX + dpl)	
FD96dpl	SUB (IY + dpl)	
97	SUB A	
90	SUB B	
91	SUB C	modifiés
92	SUB D	
93	SUB E	
94	SUB H	

95	SUB L	modifiés
D6v	SUB v	
AE	XOR (HL)	
DDAEdpl	XOR (IX + dpl)	
FDAEdpl	XOR (IY + dpl)	
AF	XOR A	
A8	XOR B	
A9	XOR C	idem que OR
AA	XOR D	
AB	XOR E	
AC	XOR H	
AD	XOR L	
EEv	XOR v	

ANNEXE 2
ASSEMBLEUR & DESASSEMBLEUR

Maintenant que vous connaissez l'ASSEMBLEUR , il faudrait vous équiper d'un logiciel **"ASSEMBLEUR"** vous permettant d'échapper au studieux travail de rentrée des codes !!

Il existe pour le moment sur le marché deux ASSEMBLEURS très corrects : **celui de la société LOGI'STICK et celui du mensuel MICRO SYSTEMES** .

Le premier est relativement bien fait mais présente **quelques lacunes** : limitations au niveau des labels , pas de directives (ORG , DEFM ...) , un peu compliqué d'emploi .

Le deuxième , écrit entièrement en langage machine , est **pratiquement parfait** . En effet , il tient sur un carte mémoire de 4 Kilo-octets , possède beaucoup de pseudo-instructions ainsi que des fonctions de controle très puissantes . Nous le préférons nettement à celui de la société LOGI'STICK pour sa rapidité , sa compacité et , surtout , pour sa grande facilité d'emploi . Tous les listings source de cet ouvrage ont d'ailleurs été réalisés avec ce programme. Il est paru dans le numéro 49 de la revue MICRO SYSTEMES .

Vous devez sûrement vous douter qu'il existe un programme qui permet d'effectuer l'opération inverse d'un ASSEMBLEUR . Effectivement , quand vous avez assemblé un programme en mémoire , vous désirez probablement vérifier si cela a été bien opéré . Afin de calmer vos angoisses , il existe un logiciel appelé **"DESASSEMBLEUR"** permettant de lister la mémoire et de désassembler les codes entrés . En général , un DESASSEMBLEUR est accompagné d'un programme **"MONITEUR"** possèdant d'autres fonctions intéressantes (mode TRACE , sauvegarde de codes , listings de codes en ASCII , sortie sur imprimante ...) .

Trois logiciels se disputent la "suprématie du marché" : **celui de LOGI'STICK , de CANON et de MICRO SYSTEMES** .

La société LOGI'STICK édite , sur la même cassette que l'ASSEMBLEUR , un DESASSEMBLEUR-MONITEUR . A l'instar du premier programme , le DESASSEMBLEUR est **très bien fait** et autorise de multiples opérations sur la mémoire . Il est rapide et très pratique d'utilisation . Actuellement , la cassette incluant l'ASSEMBLEUR et le DESASSEMBLEUR-MONITEUR est vendu au prix de **125 Francs par le CLUB C7** (Le plus bas prix constaté , réservé uniquement aux adhérents ...) .

La société CANON a commercialisé , sous forme de carte mémoire prèprogrammée , un DESASSEMBLEUR-MONITEUR **très puissant** . Cette carte **XP-140F** , dotée de 8 kilo-octets de ROM et de 4 kilo-octets de RAM utilisateur, est vendue **entre 500 et 550 francs** . Elle permet quantité de manipulations grâce à dix fonctions supplémentaires BASIC (RENUM , MERGE , AUTO ...) et à un mode MONITEUR très performant (points d'arrêts , désassemblage multiples, entrée de codes ...) .

Enfin , le journal MICRO SYSTEMES a publié dans son numéro 42 , un DESASSEMBLEUR-MONITEUR **complet** . Il s'avère un peu moins fouillé que celui de la société LOGI'STICK mais possède tout de même une puissance de travail honnête .

Pour que vous n'ayez pas de problèmes à trouver ces divers logiciels , voici les adresses des différentes sociétés les commercialisant :

CANON FRANCE
7 av. Albert EINSTEIN
Centre d'affaire
93153 Le blanc Mesnil

MICRO SYSTEMES
2 à 12 rue de Bellevue
75940 Paris Cedex 19

LOGI'STICK
Centre d'affaire
"Le Bonaparte"
93153 Le Blanc Mesnil

ANNEXE 3
LE CLUB C7

Ces quelques lignes vont vous renseigner sur le CLUB C7 , association fondée en OCTOBRE 1984 . Elle a réussi à imposer **une gazette complète et très recherchée ainsi que des services de haute compétence** (permanence technique, stages , service courrier très développé ...) . Rejoindre le CLUB C7 , c'est entrer dans le monde fantastique des "Canonnistes" qui maitrisent parfaitement leur machine préférée : **le CANON X-07** .

Le CLUB C7 : que du CANON X-07 ...

Le CLUB C7 est une association sous la loi 1901 . Son but est de faciliter la **compréhension** et le **développement** de l'ordinateur portable **CANON X-07** et d'agir comme un point de liaison entre les possesseurs du X-07 grâce à plusieurs atouts : une gazette , une programmathèque , une aide technique , une coopérative , des stages , etc ...

Le CLUB C7 a été fondé par des professionnels en informatique et des étudiants : toute personne désirant une aide , un journal , des contacts ... peut s'inscrire et bénéficier de tous les services du CLUB **gratuitement** , après avoir réglé sa cotisation .

Le CLUB C7 : "le SON du CANON" ...

La gazette intitulée **"le SON du CANON"** a une vocation bimestrielle . Elle est constituée , en moyenne , de 40 pages format A4 traitant de tous les domaines concernant le X-07 : BASIC , LANGAGE MACHINE , PROCESSEURS, HARDWARE , PROFESSIONNEL , ROM , COURRIER , ADRESSES , ESSAIS de LOGICIELS ...

"Le SON du CANON" est envoyée aux adhérents avec **une K7 C33** récapitulant tous les programmes de ladite gazette : les efforts de frappe source d'erreurs , longs et pénibles sont ainsi évités .

Le CLUB C7 : L'AIDE TECHNIQUE ...

Cette AIDE TECHNIQUE se présente de 2 façons : les permanences et le courrier .

Les permanences téléphoniques se situent tous les mardi après-midi de **14h à 18h au 371.08.00** .

Néanmoins , vous pouvez passer nous voir le mardi après-midi uniquement à l'adresse du CLUB .

Nous fournissons aux adhérents , aussi bien par courrier que de vive voix, tous les renseignements dont nous disposons sur le CANON X-07 : **ROM désassemblée , cours d'apprentissage au BASIC et L.M. , informations complètes sur le HARDWARE , adresses , etc ...**

Cette aide est appuyée par les connaissances étendues de techniciens chevronnés travaillant sur le X-07 depuis sa sortie en France : rien ne leur échappe !!

Le CLUB C7 : une PROGRAMMATHEQUE ...

Le développement de la programmathèque est lié à **la participation que lui apporte nos membres** : en effet , ce sont eux qui fournissent la majorité des programmes dont la liste est présente dans chaque gazette .

Pour encourager les programmeurs et garantir une bonne qualité des programmes , ceux-ci sont vendus à un prix modique (entre 5 et 20 Francs) et le programmeur recoit chaque année la moitié du montant total de la vente de ses programmes .

Les programmes commandés sont envoyés aux adhérents sur K7 avec une notice détaillée .

Le CLUB C7 : une COOPERATIVE ...

La coopérative a pour but principal de vendre des produits **moins chers que sur le marché** : nous disposons de beaucoup de produits et nous souhaitons un développement massif de cette section pour offrir aux adhérents le maximum d'avantages .

Le PRESIDENT du CLUB C7

André TONIC

BULLETIN d'INSCRIPTION au CLUB C7
33 AV. PHILIPPE AUGUSTE . 75011 PARIS

Je désire m'inscrire au CLUB C7 pour un an à partir de ma date d'adhésion . Je joins un chéque de **380 Francs à l'ordre du CLUB C7** (ou de **430 Francs** pour l'étranger en raison des frais postaux) .

NOM : ..... PRENOM : .....
PROFESSION : ..... AGE : .....
ADRESSE : .....
.....

PUBLICATIONS C7

1
* L.M & nombres aléatoires
* L'interface PERITEL X-720
* Kit appel, Maths,Astro de LSC
* Jeux : Magic Circus , Dactylo
* Trucs,astuces,informations
* BEEPS TRES SPECIAUX !!!

2 SPECIAL L.M.
* Auto-programmation,adresses
* PAINT : 3 fonctions graphiques
* FICHIER,ASS/DESASS de LSC
* Carte 40 Ko d'INFOSYSTEMES
* Jeu de la VIE en L.M.

3
* Le sous-processeur T6834
* Les BOOLEENS du X-07
* Les sympathisants du X-07
* Nautilus,Calc,Graphe de LSC
* Les adresses RST en folie
* COPYRIGHT en folie ...

Rapport de stage 1
* GENERALITES
* Mnémoniques
* Assembleur
* Modes d'adressage
* Exercices divers
* Tableau général

4
* Solitaire/Pentominos L.M
* Agenda (LSC) , Calc (P.S.)
* Camemberts Statistiques
* Tri de nombres en L.M
* Anti-BREAK en L.M
* Adresses Mots-Clés

5
* Traitement de textes (L.M.)
* adresses fin des mots-clés
* Progs en L.M et BASIC
* Graphe (P.S.) , Forth (LSC)
* Références croisées (L.M.)
* Instruction BOX en L.M

6
* Jeux (P.S. et LSC)
* Microbox d'ADIRIS
* Les routines de conversion
* Labyrinthe 3D , alarme ...
* 40 programmes (ETSF)
* Des surprises ...

CLUB C7

.....

NOM : .....
PRENOM : .....
ADRESSE : .....

☐ Je désire recevoir les Numéros .....
.....
de la gazette au prix unitaire de **40 F** .

☐ Je désire recevoir les Numéros .....
des rapports de stage au prix unitaire de **100 F** .

☐ Je désire recevoir les Numéros .....
.....
de la K7 accompagnant la gazette (pas de K7 n°3) au prix unitaire de **40 F** .

TOTAL : ..... + **15 Francs de frais d'envoi** : ..... Francs .

0 CHEQUE **0 MANDAT** **0 CCP** **à l'ordre du CLUB C7**

ANNEXE 4
BIBLIOGRAPHIE

Cette bibliographie , non exhaustive , peut vous permettre de vous documenter sur tel ou tel aspect de l'ASSEMBLEUR Z-80 ou du CANON X-07 .

EDITIONS	TITRE
SYBEX	"La programmation du Z-80" de R. ZACKS
EYROLLES	"L'ASSEMBLEUR FACILE du Z-80"
P.S.I.	"Programmer en ASSEMBLEUR" de A. PINAUD
CANON	Manuel de la carte MONITEUR XP-140F
CLUB C7	Toutes les gazettes "Le SON du CANON"
TESTS	L'Ordinateur Individuel
S.P.E.	Micro Systèmes

Cet ouvrage a été composé sur :

apple®

Réalisation PROMOCOM

49, rue Fondary
75015 PARIS
Tél. 45 79 80 12