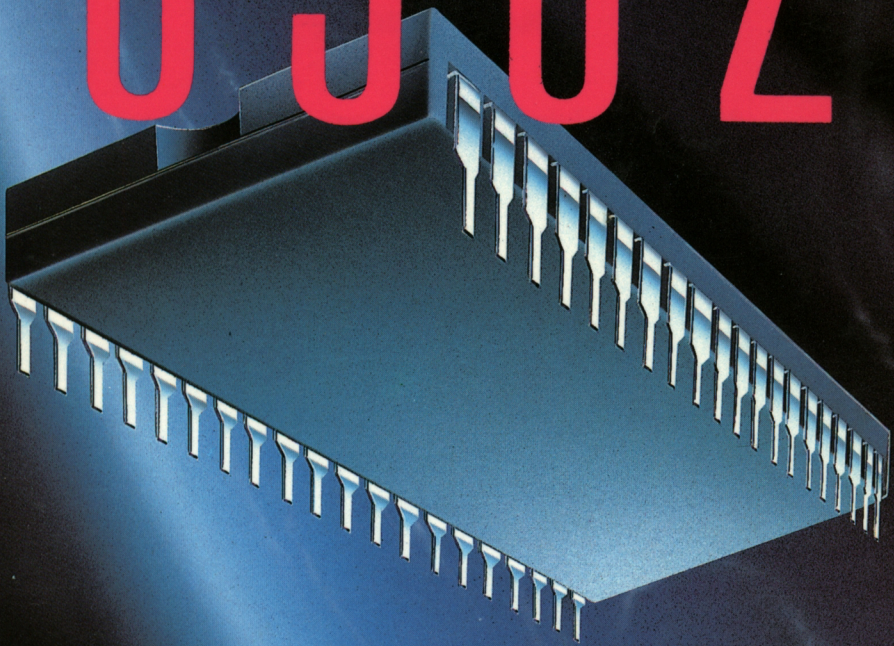
RODNAY ZAKS

APPLICATIONS DU 6502



SYBEX

APPLICATIONS DU 6502

RODNAY ZAKS

Traduction française par Daniel-Jean David

ISBN 2-7361-0032-1

(Editions originale :
ISBN 089588-015-6)

TABLE DES MATIÈRES

PRÉFACE

Le livre présente des techniques d'applications pratiques pour le microprocesseur 6502. Il suppose une connaissance des éléments de la programmation d'un microprocesseur, tels qu'ils ont été exposés dans le précédent livre de cette série (réf. C3 : Programmation du 6502). La compréhension de la programmation du microprocesseur lui-même (le 6502) n'est qu'un préalable à la programmation d'une carte micro-ordinateur connectée à des appareils réels. Le problème qui se pose ensuite est d'apprendre à écrire de véritables programmes d'application mettant en jeu les ports d'entrée-sortie et les autres éléments disponibles sur un système réel. Ce livre lui est consacré. Il présente les techniques requises par des applications caractéristiques, utilisant les boîtiers d'entrée-sortie réels disponibles sur une carte.

Les programmes présentés dans ce livre exigent, pour être mis en œuvre, un minimum de hardware. Nous recommandons au lecteur de mettre en pratique, sur un système réel, les concepts et les techniques présentés ici. Une description réaliste de cartes d'applications possibles sera présenté. Les programmes sont applicables à toute carte microordinateur basée sur un 6502, notamment, le KIM, le SYM, et l'AIM 65. De nombreux programmes fonctionnent directement sur l'une ou l'autre de ces cartes. D'autres demanderont des adaptations mineures. Néanmoins, les techniques et les concepts restent les mêmes.

Les programmes d'application présentés dans ce livre permettront au lecteur de construire un système d'alarme complet pour une habitation, comprenant un détecteur d'incendie, et bien d'autres installations : un piano électronique, un régulateur de vitesse de moteur, un contrôleur de train électrique, une horloge 24 heures, un système de commande de feux de carrefours simulés, un générateur de code Morse, une boucle de régulation industrielle de température, comprenant un convertisseur analogique-digital, etc.

Ce livre a pour but d'enseigner toutes les techniques fondamentales d'utilisation du 6502, en vue de diverses applications dans le monde réel. Il est précédé, dans la même série, de 331 : ***Programmation du 6502.***

1

INTRODUCTION

La compréhension du fonctionnement du microprocesseur proprement dit n'est que l'un des problèmes auquel le débutant se trouve confronté (1). Il convient ensuite d'apprendre à programmer de façon effective, en utilisant les organes d'entrées-sorties connectés à la carte micro-ordinateur. Tel est le but de ce livre. Il est naturellement impossible de traiter de tous les composants. Nous avons donc opéré un choix parmi les composants d'entrée-sortie les plus importants habituellement connectés à un 6502. Les programmes présentés ici conviendront probablement à la majorité des applications.

Premier programme proposé : la commande d'un PIO, boîtier d'entrées-sorties parallèles. Vous apprendrez à utiliser la scrutation et les interruptions, à générer des impulsions, à mesurer des délais et à commander des organes réels d'entrée-sortie tels que des interrupteurs ou des relais, et même des organes plus complexes, comme un convertisseur analogique-digital et un moteur, entre autres. Vous apprendrez aussi à vous servir de boîtiers plus élaborés : les *temporisateurs programmables,* par exemple. Nous vous présenterons, en outre, des interfaces à des périphériques simples, de sorte que vous puissiez monter une carte d'applications, et faire des exercices pratiques.

Une seule solution pour apprendre véritablement à programmer, pour devenir un programmeur efficace : la pratique. Comment pratiquer ? Il faut, en premier lieu, disposer d'une carte micro-ordinateur : le KIM, le SYM, l'AIM 65, ou toute autre carte basée

(1) Nous abordons cette question dans notre livre 331 : ***Programmation du 6502.***

sur le 6502. Toutes ces cartes possèdent au moins un PIO (souvent 2), et au moins 2 temporisateurs (quelquefois plus), et devraient donc pouvoir être utilisées, avec éventuellement des adaptations de détail, pour tous les programmes présentés dans ce livre.

Le matériel supplémentaire dont vous aurez besoin pour faire fonctionner certains programmes sera évoqué dans les chapitres IV, V, et VI. Il se réduit à un minimum, que vous vous procurerez facilement. En particulier, vous trouverez aux chapitres mentionnés la description de cartes d'applications, que nous vous suggérons de construire à partir de composants répandus et bon marché. Cela vous permettra d'exécuter, sur une carte, les programmes de chaque chapitre à l'aide de votre micro-ordinateur, et de la carte d'applications. Il est conseillé de ne pas négliger de construire ces cartes, afin d'acquérir de la pratique.

Ce n'est, certes, pas absolument indispensable. Il est possible d'apprendre les techniques de base à la seule lecture du livre. Mais si vous souhaitez aller plus loin, alors la pratique effective est vivement recommandée.

Liaison de votre microprocesseur avec le monde réel

Relier un microprocesseur au monde réel implique d'abord de construire une carte micro-ordinateur élémentaire, puis de la connecter à des périphériques proprement dits. Ce livre présentera en détail à la fois les composants matériels et les programmes appropriés aux unités les plus fréquemment utilisées. Pour concevoir des programmes industriels mettant en jeu des appareils coûteux, comme des feux de carrefours, on utilisera, sur les cartes d'applications, des composants capables de les simuler comme les LED. Appliquer le programme à un système de carrefours réels nécessite uniquement de changer le hardware de l'interface. Le programme reste identique. Par suite, les techniques exposées ici seront applicables dans la vie.

La pédagogie

Chaque programme sera présenté en détail. Nous aborderons, tour à tour, son but, son ordinogramme, l'interface hardware, les périphériques, le programme lui-même et l'analyse complète des techniques utilisées. Chaque chapitre se suffit pratiquement à lui-même. Par exemple, il n'est pas nécessaire de comprendre toutes les caractéristiques du PIO du chapitre II pour lire le chapitre III. Une lecture dans l'ordre facilite, néanmoins, une bonne compréhension.

Le chapitre II introduit tous les boîtiers d'entrées-sorties parallèles utilisés dans un système à 6502, du 6520 au 6532. On sait que toutes les cartes basées sur le 6502 utilisent actuellement ces boîtiers standard. Si vous n'êtes pas familiarisés au maniement de ces boîtiers, la lecture de ce chapitre est donc indispensable.

Le chapitre III présente la « carte 6502 standard » et quelques variantes bien connues : le KIM, le SYM, l'AIM 65 (il en existe d'autres). La plupart des exemples présentés dans ce livre sont prévus pour le KIM ou le SYM, et passent sur les autres cartes avec des changements mineurs.

Le chapitre IV introduit les techniques de base nécessaires pour connecter des composants simples : relais, interrupteurs, haut-parleurs. On utilisera la première carte d'applications pour construire, par exemple, un générateur Morse ou un composeur de numéros de téléphone.

Le chapitre V présente des applications domestiques ou industrielles plus complexes. On utilisera la seconde carte d'applications pour simuler des feux de carrefour, construire un système d'alarme antivol, un convertisseur analogique-digital ou un piano électronique.

Le chapitre VI réalise la connexion d'une carte micro-ordinateur à de véritables périphériques bon marché : lecteur de ruban perforé, clavier ou imprimante.

Enfin, le chapitre VII établit un résumé et une synthèse.

Pour vous aider à développer ces programmes complexes, pour lesquels un assembleur est indispensable, vous trouverez dans l'appendice A un assembleur 6502 complet, écrit en BASIC.

A la page suivante se trouve un formulaire standard de programmation, pour vous aider à écrire vos programmes sur 6502.

SYBEX **STANDARD PROGRAMMING FORM**

PROGRAM _____

AUTHOR _____ DATE __/__/__ SHEET __/__

HEXADECIMAL				SYMBOLIC ASSEMBLER INSTRUCTIONS			
ADDRESS	1	2	3	LABEL	MNEM	OPERAND	COMMENT

Fig. 1-1 : Formulaire standard de programmation.

2

LES BOITIERS D'ENTRÉES-SORTIES

INTRODUCTION

Pour connecter des organes d'entrée-sortie variés à une carte 6502, en vue de réaliser des applications pratiques, il est essentiel de comprendre les ressources des systèmes 6502 dans le domaine des entrées-sorties. Le lecteur peu familiarisé avec les termes de base ou les techniques élémentaires, comme la « scrutation », est encouragé à se reporter au livre précédent de la même série (réf. **331** : *Programmation du 6502).*

Nous allons, dans ce chapitre, passer systématiquement en revue les boîtiers d'entrées-sorties parallèles, utilisés sur presque toutes les cartes 6502 pour créer les possibilités d'entrées-sorties nécessaires. Il est indispensable de comprendre, au moins, le fonctionnement d'un « PIO » comme le 6522 avant de lire les chapitres d'applications. Les détails du fonctionnement des temporisateurs ou d'autres dispositifs particuliers (comme le registre à décalage) ne sont pas essentiels, pour une première lecture. De même, il n'est pas important de retenir les formats détaillés des différents registres internes des 6520, 6522, 6530 et 6532. Ils sont fournis ici à titre de référence pour les chapitres suivants.

Nous vous suggérons donc de lire avec soin l'une des sections sur un PIO comme le 6520 ou le 6522, sans essayer de retenir tous les détails mais en prêtant attention au fonctionnement. Presque toutes les applications utilisent un PIO, c'est-à-dire l'un des boîtiers présentés dans ce chapitre.

En plus de ces boîtiers, la plupart des cartes micro-ordinateurs possèdent d'autres interfaces spécialisées : interface cassette ou écran. Le lecteur intéressé se reportera, pour plus de détails, aux notices des constructeurs ou au livre C5 *(Techniques d'interface aux microprocesseurs).*

DÉFINITIONS DE BASE

Cette section est un rappel des termes utilisés dans le présent chapitre.

Les trois dispositifs d'entrées-sorties essentiels, utilisés sur pratiquement tous les micro-ordinateurs, sont le « PIO », l' « UART » et le « temporisateur ». Examinons-les.

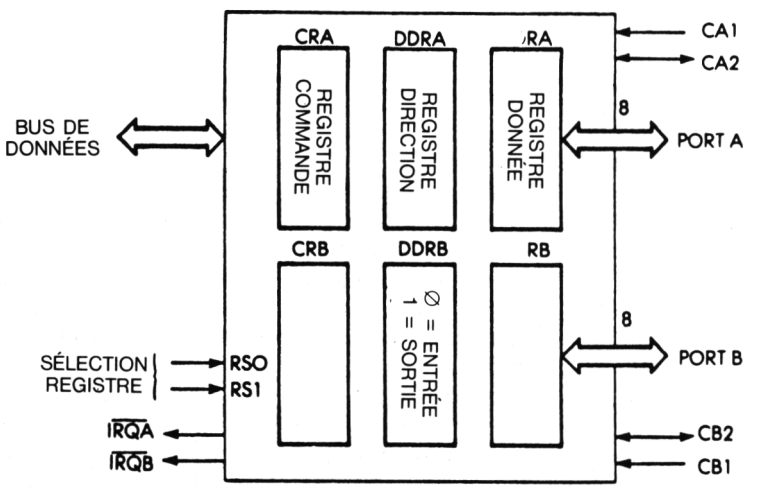


Fig. 2-1 : PIO type

Le PIO

Le « PIO » ou « Parallel Input-Output chip » (boîtier d'entrées-sorties parallèles) est un composant qui procure au moins deux ports de 8 bits en parallèle. La direction d'utilisation des lignes de chaque port y est programmable individuellement. La direction de chaque ligne est généralement déterminée par le contenu d'un registre « direction » associé à chaque port. Par exemple, lorsqu'un bit donné du registre « direction » est « 0 », la ligne correspondante

du port sera une entrée. Avant d'utiliser le PIO, le programmeur devra d'abord charger le registre direction de chaque port, pour définir le sens de fonctionnement des différentes lignes. Certaines contraintes de détail peuvent être imposées par les constructeurs notamment la programmabilité de la direction par groupes de 4 bits, ou l'attribution de fonctions spéciales à certains bits (1). Nous en rencontrerons quelques-unes sur les boîtiers présentés dans ce chapitre. Le schéma général du « PIO standard » apparaît figure 2-1. Les deux tampons pour les ports A et B apparaissent à droite de la figure, et les registres direction associés à chaque port à gauche des tampons. On observera, en outre, sur ce schéma simplifié, la présence de deux registres de commande. Le registre de commande est nécessaire pour gouverner le fonctionnement des signaux de commande à l'œuvre dans ce PIO. Il doit spécifier la procédure dite de « poignée de main », et déterminer si les signaux de commande vont positionner des indicateurs, ou déclencher des interruptions. De la même façon, il tranchera la question de savoir si la transition considérée a lieu de haut en bas, ou de bas en haut. En principe, le programmeur devra spécifier le contenu du registre avant toute utilisation des lignes de commande du composant, et examiner le contenu du registre de commande pour déterminer si une interruption interne ou toute autre condition particulière a été détectée (rôle de registre d'état).

En plus des deux ports de données, le PIO doit aussi posséder des lignes de commande, qui permettent une « poignée de main » automatique avec un périphérique. Les lignes de commande sont visibles à droite du PIO standard de la figure 2-1, et s'intitulent respectivement CA1, CA2 pour le port A, et CB1, CB2 pour le port B.

A titre d'exemple d'une procédure de « poignée de main », le périphérique pourrait fournir un signal « donnée prête » sur CA1. Le microprocesseur répondrait alors par un signal « donnée demandée » sur CA2. De surcroît, la réception de « donnée prête » sur CA1 doit être marquée dans le registre de commande. On pourrait générer une demande d'interruption pour alerter le 6502 à la suite de cet événement. Il s'agit là d'un exemple simple caractéristique de la séquence nécessaire à une synchronisation effective des échanges. La plus grande partie de cette procédure est automatique, à l'intérieur du PIO standard, et les options sont définies par le contenu du registre de commande. Les détails spécifiques à chaque boîtier seront présentés à partir de la section suivante.

(1) En particulier les bits 6 et 7.

Le temporisateur

Un impératif fondamental : dans la plupart des applications, les programmes doivent être capables de générer des *délais* déterminés. Les délais peuvent être produits par des techniques software, ou par hardware, à l'aide de temporisateurs. Tant qu'on ne fait pas appel à des interruptions dans le système, il est commode de générer les délais par des boucles programmées (voir la réf. C3 pour les détails). Néanmoins, dans des cas plus compliqués, ou en cas d'interruptions, il est préférable d'utiliser un ou plusieurs temporisateurs hardware.

Utilisation du temporisateur en sortie

Sous sa forme la plus simple, un temporisateur est un registre compteur (à 8 ou 16 bits). En mode sortie, on charge, par programme, le registre du temporisateur avec une valeur donnée. On donne alors un signal de départ, et le comptage commence. La plupart des temporisateurs utilisent l'horloge du système (habituellement 1 MHz, soit des impulsions de 1 μs) mais ce n'est pas une nécessité. Le nombre présent dans le compteur sera décrémenté de 1 à chaque impulsion ; si la valeur placée dans le compteur était N, le contenu du compteur arriverait à 0 après N impulsions, c'est-à-dire N microsecondes (dans le cas d'une horloge à 1 MHz.). Lorsque le compteur arrive à 0, un signal est généré, entraînant soit le positionnement d'un indicateur d'état dans le boîtier temporisateur, soit la génération d'une interruption externe. En fonction de la précision requise, le programme va scruter les temporisateurs, ou procéder par interruptions. Nous présenterons des programmes-type dans la suite de ce chapitre.

Si le temporisateur n'était muni que d'un registre 8 bits, il ne pourrait compter que jusqu'à 256. Avec une horloge standard, le délai maximum serait de 256 microsecondes : trop court pour la plupart des applications. Naturellement, on pourrait utiliser l'interruption produite à la fin des 256 μs pour mettre à jour un emplacement mémoire, puis tester si cette case mémoire a atteint une valeur donnée. Le procédé est, toutefois, fastidieux, et la mesure peu précise. Un temporisateur muni d'un registre 8 bits est donc insuffisant. On utilisera, de préférence, deux techniques, dont la plus simple, conceptuellement est d'utiliser pour le compteur un registre 16 bits. Il est alors possible de compter jusqu'à 64 K, donc de 1 à 65 536 microsecondes (à peu près 65 millisecondes). C'est, certes, suffisant pour la plupart des applications, mais cette technique exige que le timer soit chargé en deux fois, au moins,

puisque le bus de données n'a que 8 bits de large. Le programme doit d'abord charger une moitié du registre, puis l'autre. D'où un manque de commodité certain.

L'autre technique pour générer des délais de valeurs très différentes consiste à utiliser des circuits de division internes au temporisateur, lequel apparaîtra alors à l'utilisateur comme formé de, disons, quatre registres. Par exemple, si on utilise le premier registre, le délai généré s'exprimera en périodes d'horloge (typiquement 1 μs). Avec le second registre, l'unité de délai sera de 8 cycles d'horloge ; avec le troisième, elle sera de 64 cycles d'horloge et avec le quatrième, de 1 024 cycles (soit 1 milliseconde avec une horloge à 1 MHz). Cette approche est un peu plus commode pour le programmeur. Elle permet de charger le temporisateur en une seule opération, et de générer des délais très variables. Inconvénient : elle augmente la complexité interne du composant.

Utilisation du temporisateur en entrée

On peut utiliser un temporisateur en entrée pour mesurer la durée d'une impulsion extérieure, ou l'intervalle entre deux impulsions successives. Dans ce cas, le contenu initial du temporisateur est 0, et le compteur s'incrémente à chaque intervalle de temps. Lorsque le délai a été mesuré, le composant positionne un indicateur, ou génère une interruption. Le programme est responsable de la lecture du contenu du registre de comptage, qui marque la durée de l'événement extérieur.

Train d'impulsions

On peut utiliser un temporisateur, non seulement pour générer ou mesurer une impulsion, mais aussi pour générer ou compter des trains d'impulsions. Dans le cas d'une seule impulsion, on dit qu'on utilise le temporisateur en mode « monostable ». Pour un train d'impulsions, on parle du mode « oscillateur ». Un certain nombre d'options peuvent, éventuellement, spécifier si le temporisateur sera activé ou arrêté sur un front montant ou descendant du signal, ou si on sera sensible à des niveaux plutôt qu'à des fronts. La synchronisation et la valeur logique des indicateurs peuvent également être spécifiées. Enfin, les conditions d'établissement et de remise à zéro de l'indicateur d'état sont habituellement programmables. En raison du grand nombre de variantes possibles, chaque temporisateur tend à avoir une « personnalité » bien à lui. Il faut l'étudier en détail avant de l'utiliser.

L'UART

UART est l'abréviation de « Universal Asynchronous Receiver and Transmitter » (émetteur-récepteur asynchrone universel). Sa fonction essentielle est d'effectuer des conversions série-parallèle et parallèle-série. L'UART standard présente un certain nombre d'options généralement requises pour les communications série, telles que la parité (vérification, inhibition ou génération) et le nombre de bits de départ et d'arrêt. La conversion est effectuée à l'aide d'un registre à décalage interne, qui peut être incorporé dans certains boîtiers d'entrées-sorties.

Les boîtiers d'entrée-sortie utilisables avec le 6502

Pratiquement, toute carte basée sur le 6502 nécessite au moins 2 PIO et un temporisateur. Ces fonctions seront généralement remplies par une combinaison de 6520 et 6530, ou de 6522 et 6532. Les premiers ont été introduits, au départ, par MOS Technology. Le 6502 est maintenant fabriqué par plusieurs constructeurs, tels que Synertek et Rockwell. De nouveaux boîtiers annexes sont venus s'ajouter à la famille : le 6522 et le 6532, par exemple. Plusieurs autres seront probablement introduits à l'avenir.

Il reste qu'actuellement les boîtiers les plus importants sont le 6520, le 6530, le 6522 et le 6532. Nous allons maintenant les décrire.

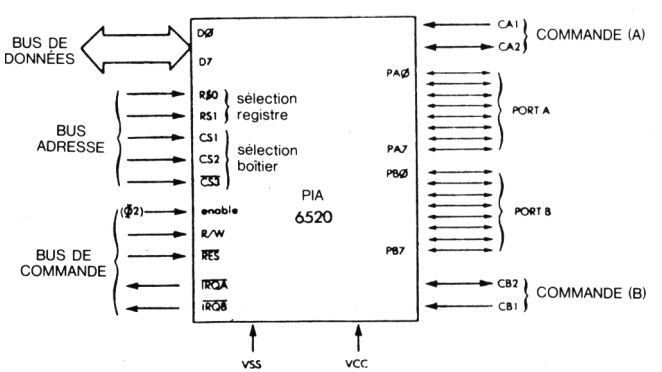


Fig. 2-2 : Le PIA 6520

LE 6520 (PIA)

Le 6520 est pratiquement un « PIO » standard, tel que nous l'avons défini. Il a été conçu pour être compatible, broche pour broche, avec le 6820 de Motorola. Son constructeur le désigne sous l'intitulé de PIA (« Peripheral Interface Adapter » = adaptateur d'interface périphérique). Les signaux du 6520 sont sur la figure 2-2. Son architecture interne est présentée par la figure 2-3.

En se reportant à la figure 2-3, on constate que ce composant possède deux ports d'entrées-sorties : A et B. Chaque port possède un tampon. A et B ne sont cependant pas rigoureusement identiques. Le tampon fonctionne, en fait, comme tampon de sortie, pas d'entrée. Chaque port dispose d'un registre direction (« DDR »), qui spécifie la direction de chaque ligne du port. Une valeur « 0 » dans ce registre indique une entrée ; un « 1 » indique une sortie. Cette convention résulte de considérations de sécurité, puisque lorsqu'on effectue un « RESET », le contenu de tous les registres (registre direction compris) est mis à 0. Il en résulte que toutes les lignes se trouvent configurées en entrée ; c'est la manière sûre de

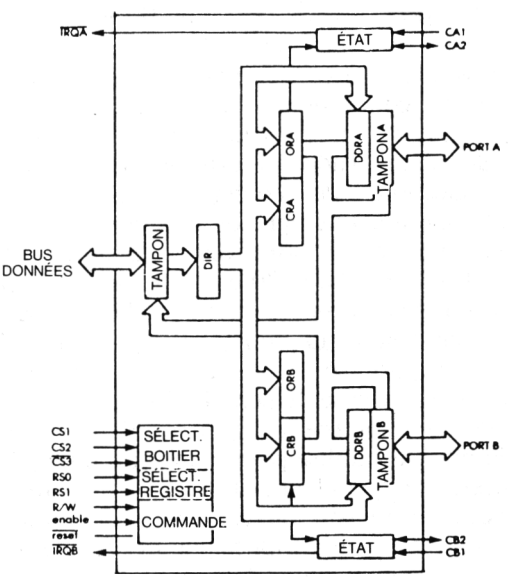


Fig. 2-3 : Architecture interne du 6520

démarrer un système : aucune impulsion externe ne pourra être générée avant le début de l'exécution du programme.

Chaque port possède deux registres : *commande* et *sortie*. Les données envoyées par le 6502 vers le PIA sont dirigées vers *le registre de sortie* (ORA ou ORB) du port concerné, et elles y sont conservées. Le fonctionnement du *registre de commande* (CRA ou CRB) est expliqué ci-dessous. Rapidement, cet élément spécifie les différentes options de commande et contient les informations d'état de chaque port.

Chaque port possède deux lignes de commande externe appelées CA1 et CA2, pour le port A. CA1 est unidirectionnelle : du périphérique vers le 6520. CA2 est bidirectionnelle. Elle peut être utilisée en entrée ou en sortie.

Les deux ports sont, du point de vue logique, équivalents et symétriques. Mais ils comportent, en pratique, quelques différences. Le port B peut, en particulier, fournir des courants supérieurs à ceux du port A, et les signaux de commande ne jouent pas un rôle totalement symétrique.

La figure 2-2, et la partie gauche de la figure 2-3, montrent que le bus de données relie le tampon interne du 6520 au bus de données du système. Le boîtier peut générer deux demandes d'interruption (respectivement IRQA et IRQB), si le contenu des registres de commande des ports A et B le spécifie. Enfin, il convient de positionner trois entrées de sélection de boîtier CS1, CS2 et $\overline{CS3}$. Cette conception a été élaborée par Motorola pour permettre de connecter facilement jusqu'à 8 boîtiers, sans faire appel à un décodeur d'adresse externe. En pratique, le grand nombre de sélections de boîtier est peut-être cause d'un inconvénient (manque d'une ligne de sélection de registre) (1). Deux lignes de sélection de registre sont connectées au bus adresse. On les désigne par RS0 et RS1. Le 6520 est donc perçu du programmeur comme *quatre* emplacements-mémoire. Cela peut sembler surprenant puisque nous venons de voir (cf. fig. 2-3) que l'on dénombre trois registres par port, soit un total de *six* registres. Comment accéder à 6 registres avec seulement 4 adresses ? Le problème vient de la limitation du nombre de broches. Un bit du registre de commande, le bit 2, est utilisé pour établir un multiplex entre deux jeux de registres. Lorsque le bit 2 du registre de commande est égal à « 0 », on accède au registre direction du port considéré. Lorsqu'il est égal à « 1 », le tampon périphérique est sélectionné.

On dispose de 3 autres lignes de commande : « R/$\overline{W}$ » (lecture-écriture), « horloge » (qui reçoit habituellement le signal φ 2) et « reset ».

(1) Voir plus loin.

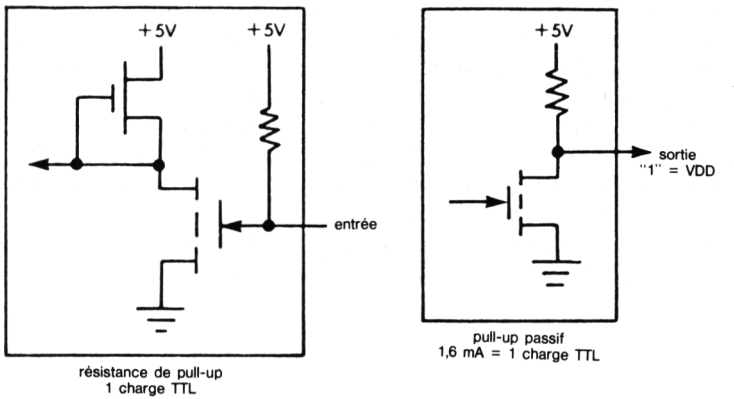


Fig. 2-4 : Tampon A

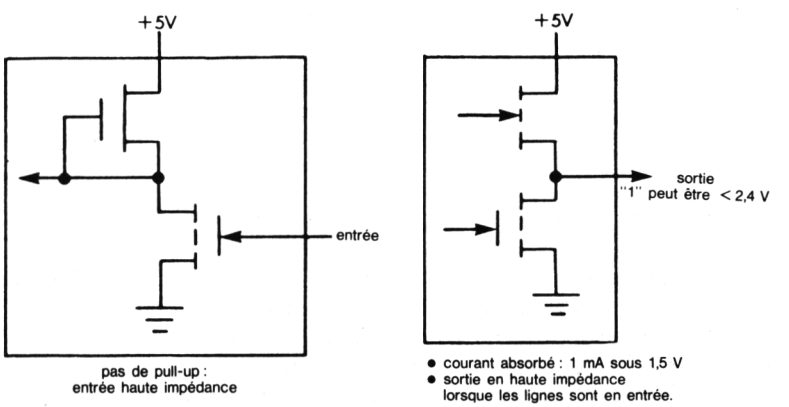


Fig. 2-5 : Tampon B

Différences entre le port A et le port B

Bien que logiquement équivalents, le port A et le port B sont physiquement différents. Les tampons du port A ont des *pull-ups passifs*. Ils peuvent absorber 1,6 mA, ce qui les rend capables de

commander une charge TTL standard. Sur le port B, les tampons fonctionnent en *push-pull* (cf. fig. 2-4 et 2-5). Comme il s'agit de dispositifs *actifs,* la tension correspondant à l'état logique « 1 » peut être inférieure à 2,4 V (contre V_{DD} dans le cas du port A). Ils peuvent cependant fournir un courant supérieur (1mA sous 1,5 V), de sorte qu'il devient possible de les relier directement à des LED ou à des Darlington. Lorsque le port B est utilisé comme entrée, le tampon de sortie passe à l'état *haute impédance.* L'entrée présentera alors une impédance supérieure à 1 Mégohm. La figure 2-4 montre le détail du port A et la figure 2-5 le détail du port B.

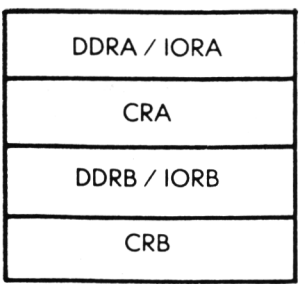


Fig. 2-6 : Carte d'implantation mémoire du 6520

Les registres internes

Considérons maintenant, plus en détail, les ressources spécifiques et les particularités du 6520. Nous avons déjà noté que le 6520 dispose de 6 registres internes : les deux tampons (qui ont la même adresse que les registres de sortie), les deux registres direction et les deux registres de commande. Cependant, en raison de la limitation du nombre de broches, il n'existe que deux lignes de sélection de registre, appelées respectivement RS0 et RS1. La carte d'implantation mémoire qui en résulte (fig. 2-6) montre que, par exemple, les registres DDRA (Direction port A) et IORA (E/S port A) partagent la même adresse logique. Le 6520 fait la différence, de façon interne, entre DDRA et IORA à l'aide de la valeur du bit 2 du registre de commande. La sélection des registres est présentée figure 2-7. Quand le bit 2 du registre de commande (CRA-2 ou CRB-2) est à 0, DDR est sélectionné. Quand il est à 1, c'est IOR (registre d'E/S). Le registre de commande est le seul qui puisse être adressé directement par RS0 et RS1, puisqu'il est logiquement

nécessaire de préciser son contenu avant d'accéder aux autres registres.

RS1	RS0	CRA-2	CRB-2	REGISTRE SÉLECTIONNÉ
0	0	1	–	E/S A
0	0	0	–	DDRA
0	1	–	–	CRA
1	0	–	1	E/S B
1	0	–	0	DDRB
1	1	–	–	CRB

Fig. 2-7 : Sélection des registres dans le 6520

Cette technique présente deux inconvénients : 1° la complication excessive de l'initialisation du composant, et 2° la nécessité d'insérer des instructions supplémentaires pour modifier, à chaque fois, le bit 2 de CRA, dans l'hypothèse où le programme a besoin d'accéder successivement à DDRA et à IORA.

Le registre de commande

Le contenu du registre de commande est explicité par la figure 2-8. Nous avons déjà souligné que le bit 2 de ce registre joue un rôle particulier : il fait la différence entre les registres DDR et IOR du port considéré. Les autres bits du registre déterminent les options des deux lignes de commande disponibles sur chaque port. Deux bits servent d'indicateurs d'état ou d'interruption. Les fonctions de CA2 et CB2 sont déterminées par les bits 3, 4 et 5 du registre de contrôle correspondant. Elles sont développées figures 2-9 et 2-10.

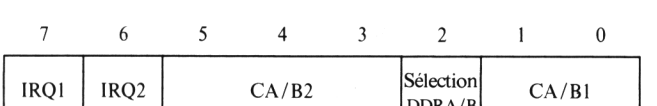


Fig. 2-8 : Registres de commande du 6520

CRA BIT 5	4	3	MODE	EFFET
1	0	0	Poignée de main en lecture	• La transition d'interruption sur l'entrée CA1 met CA2 à 1. • L'instruction de lecture du port A met CA2 à 0.
1	0	1	Sortie d'impulsion	• La lecture du port A met CA2 à 0 pendant 1 cycle (= acquittement vers le périphérique).
1	1	0	Sortie manuelle	• Met CA2 à 0
1	1	1	Sortie manuelle	• Met CA2 à 1.

Fig. 2-9 : Commande de CA2 sur le 6520

CRB BIT 5	4	3	MODE	EFFET
1	0	0	Poignée de main en écriture	• La transition d'interruption sur l'entrée CB1 met CB2 à 1. • L'écriture sur le port B met CB2 à 0.
1	0	1	Sortie d'impulsion	• L'écriture sur le port B met CB2 à 0 pendant 1 cycle (= acquittement vers le périphérique).
1	1	0	Sortie manuelle	• Met CB2 à 0
1	1	1	Sortie manuelle	• Met CB2 à 1.

Fig. 2-10 : Commande de CB2 sur le 6520

Les bits 0 et 1 commandent les entrées CA1 et CB1 comme le montre la figure 2-11.

CR BIT 1	CR BIT 0	TRANSITION ACTIVE DU SIGNAL D'ENTRÉE	SORTIE IRQ
0	0	Négative	Désactivée (à 1 en permanence).
0	1	Négative	Activée (passera à 0 lorsque le bit 7 de CR sera mis à 1 par la transitin sur CA1/CB1).
1	0	Positive	Désactivée.
1	1	Positive	Activée (comme ci-dessus).

Fig. 2-11 : Commande des interruptions (entrées CA1 et CB1)

Utilisation du 6520

Après un « Reset », le contenu de tous les registres est zéro. Il faut donc d'abord initialiser le 6520, pour spécifier les configurations d'entrées ou sorties de chaque port. Il convient aussi de spécifier les options du registre de commande, registre dans lequel le bit 2 doit normalement être laissé à 1, pour permettre au 6502 d'accéder ensuite directement au registre IOR.

Une séquence type pourrait être :

```
LDA   #$0F      "00001111" = 4 ENTRÉES,
                4 SORTIES
STA   DDRA      CONFIGURE
                LA DIRECTION
LDA   #COM      OPTIONS DE COMMANDE
                DONT BIT 2 = 1
                POUR ADRESSER IORA
STA   CRA
```

Entrées-sorties

L'envoi de données sur le port A se fera par l'intermédiaire des deux instructions suivantes (en supposant que CRA-2 = "1")

```
LDA   #DONNÉE   OU BIEN LDA $20
                (DONNÉE EN MÉMOIRE)
STA   IORA
```

La lecture d'une entrée connectée au 6520 se fera par :

```
LDA   IORA
STA   $20        SAUVEGARDE EN MÉMOIRE
```

On sauve immédiatement le contenu de l'accumulateur dans la case-mémoire d'adresse 20 en hexadécimal. Cette ligne n'est, toutefois, pas indispensable. Dans de nombreux cas, on lira simplement le contenu de IORA dans l'accumulateur, puis on testera, probablement, sa valeur, sans le stocker en mémoire.

Précautions avec le 6520

En plus des différences entre le port A et le port B, il faut garder à l'esprit certaines caractéristiques particulières des fonctions de commande. Et notamment celle-ci : les bits 6 et 7 de CRA ou CRB sont remis à 0 si CA/B2 est une entrée, et qu'on lit CR. De même une lecture du registre de données annule le bit 7.

La poignée de main sur CB2 fonctionne pour *écrire* des données, tandis que sur CA2, il s'agit d'une *lecture*. Enfin, les bits 6 et 7 peuvent provoquer une interruption.

Scrutation de plusieurs 6520

La manière la plus simple de scruter plusieurs 6520 est de tester l'état des bits 6 et 7 du registre de commande. Lorsque les 2 bits sont à 0, le périphérique ne réclame aucun service. Si l'un ou l'autre est à 1, cela signifie qu'il y a eu une interruption interne. Il faut alors la traiter.

Technique n° 1

Pour établir rapidement lequel des quatre boîtiers a demandé qu'on s'occupe de lui, on peut utiliser une technique d'accès séquentiel en mémoire, dès lors que les adresses-mémoire des 4 registres sont placées à des adresses consécutives. L'adresse en sera attribuée à CRA1, $n + 1$ à CRBA, $n + 2$ à CRA2, $n + 3$ à CRB2, etc. Le programme peut alors utiliser l'adressage indexé indirect tel qu'il apparaît ci-dessous :

```
DÉPART   LDX   #8
SUIV     LDA   (BASE - 1, X)ACCÈS AU CR SUIVANT
         BMI   SERVICE      IRQ DEMANDÉE
         DEX                X = X - 1
         BEQ   DÉPART
         BNE   SUIV
```

```
BASE    . WORD CRA1     PIO N01     PORT A
        . WORD CRB1                 PORT B
        . WORD CRA2     PIO N02     PORT A
        . WORD CRB2                 PORT B
        . WORD CRA3     PIO N03     PORT A
        . WORD CRB3                 PORT B
        . WORD CRA4     PIO N04     PORT A
        . WORD CRB4                 PORT B
```

Fig. 2-12 : Identification du PIO

Le registre index X reçoit la valeur initiale 8. Il sera décrémenté à chaque fois qu'on parcourt la boucle de scrutation. L'accumulateur est chargé, en premier, avec le contenu du dernier élément de la table.

```
        LDA   (BASE - 1, X)
```

Si le bit 7 était à 1 (le bit 7 est le bit de signe, soit l'indicateur N), un branchement aurait lieu vers la routine de service :

```
        BMI   SERVICE
```

Dans le cas contraire, X est décrémenté, et on teste le prochain CR :

```
        DEX
        BEQ   DÉPART    REVIENT A DÉPART
                        SI X = 0
        BNE   SUIV      VA AU SUIVANT SI X ≠ 0
```

Amélioration : L'échange des deux dernières instructions accélérerait-elle le programme ?

Technique n° 2

Il faut tester deux bits d'état dans chaque CR : aux positions 6 et 7. L'instruction « BIT » du 6502 a été créée spécialement pour cela. C'est un test non destructif qui porte sur les deux bits 6 et 7.

Le programme apparaît alors figure 2-13 :

```
        BIT   CRA
        BMI   IRQA7
        BVC   PASA
IRQA6   ....          INTERRUPTION DE CA2
                      (bit 6)
IRQA3   ....          INTERRUPTION DE CA1
                      (bit 7)
PASA    BIT   CRB     MEME CHOSE POUR LE
                      PORT B
        BMI   IRQB7
        BVC   SUIV
IRQB6   ....          INTERRUPTION DE CB2
                      (bit 6)
IRQB7   ....          INTERRUPTION DE CB1
                      (bit 7)
SUIV    BIT   ....    6520 SUIVANT
```

Fig. 2-13 : Identification des ports

L'instruction BIT est utilisée pour tester si les bits 6 ou 7 sont à 1 :

```
BIT   CRA
```

Elle agit sur les indicateurs V et N qu'on peut maintenant tester :

```
BMI   IRQA7   BIT 7 = 1
BVC   PASA    PAS D'INTERRUPTION
```

S'il s'avère qu'aucun des indicateurs n'est à 1, on se branche alors à PASA, où on teste CRB. BMI teste le bit 7. Si le bit 7 est à 1, le bit de signe est, dans ce cas, mis à 1, et on exécute la routine à l'adresse IRQA7.

Si le bit 6 est à 1, on exécute la routine à l'adresse IRQA6 qui suit le BMI.

Cette séquence peut être répétée pour n'importe quel nombre de 6520. Notez que cette procédure donne une plus grande priorité au bit 7 qu'au bit 6.

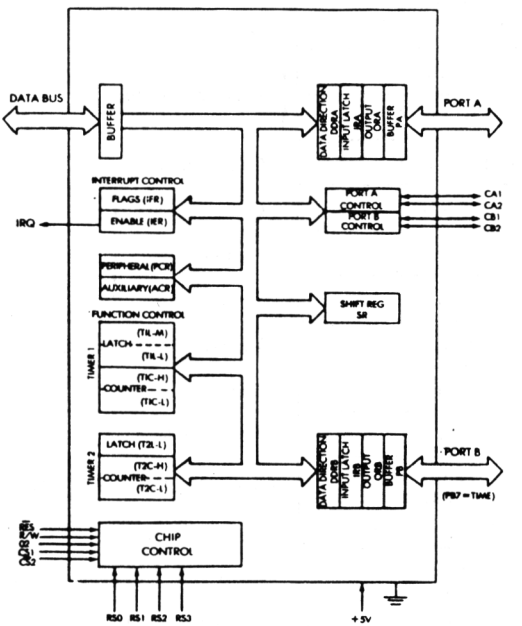


Fig. 2-14 : Architecture interne du 6522

LE 6522

Le 6522, introduit par MOS Technology, et également fabriqué par Rockwell et Synertek, est le successeur du 6520.

Le boîtier 6522, appelé VIA (Versatile Interface Adapter = Adaptateur d'interface multi-usages) est une combinaison de PIO, de temporisateur, et de registre à décalage. Il est muni de 16 registres internes (voir fig. 2-14). La carte mémoire correspondante se trouve sur la figure 2-15.

On peut distinguer quatre ensembles de registres, au point de vue fonctionnel :

1° Les registres du PIO (adresses 0 à 3, plus l'adresse F) ;
2° Les registres des temporisateurs (2 temporisateurs, adresses 4 à 9) ;
3° Le registre à décalage (adresse A) ;
4° Les registres de commande (adresses B à E).

Nous allons maintenant examiner ces quatre ensembles en détail, afin d'expliquer les potentialités du 6522.

00	ORB (PB0 A PB7)	Données d'E/S port B
01	ORA (PAO A PA7)	Données d'E/S port A avec poignée de main
02	DDRB	Registres direction
03	DDRA	
04	T1 L-L/T1C-L	Compteur bas
05	T1C - H	Compteur haut — Temporisateur 1
06	T1L-L	Tampon bas
07	T1L-H	Tampon haut
08	T2L-L/T2C-L	Tampon/compteur bas — Temporisateur 2
09	T2C-H	Compteur haut
0A	SR	Registre à décalage
0B	ACR	Registre de commande auxiliaire
0C	PCR (CA1, CA2, CB1, CB2)	Registre de commande périphérique
0D	IFR	Indicateurs — Commande d'interruption
0E	IER	Validation — Commande d'interruption
0F	ORA	Données d'E/S port A sans poignée de main

Fig. 2-15 : Carte d'implantation mémoire du VIA 6522

RS3	RS2	RS1	RS0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

00	ORB	+ handshake	PARALLEL I/O
01	ORA		
02	DDRB		
03	DDRA		
04	T1L-L(W)/T1C-L(R)	+ clear T1 Int Flag (R)	TIMER T1
05	T1C-H(R)/T1L-H + T1C-H(W)	+ T1C-L T1L-L + clear T1 Int Flag (R)	
06	T1L-L		
07	T1L-H	+ clear T1 Int Flag(W)	
08	T2L-L(W)/T2C-L(R)	+ clear T2 Int Flag(W)	TIMER T2
09	T2C-H	+ T2C-L T2L-L + clear T2 Int Flag(W)	
0A	SR		SHIFT
0B	ACR		CONTROL
0C	PCR		
0D	IFR		
0E	IER		
0F	ORA	no handshake	PARALLEL I/O

Fig. 2-16 : Registres du 6522

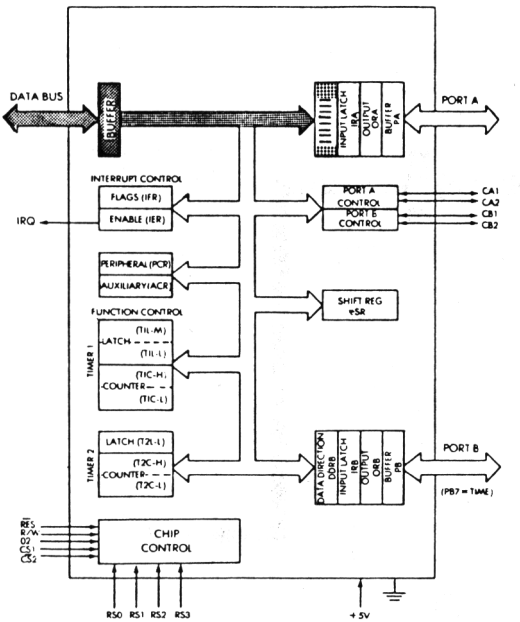


Fig. 2-17 : Utilisation du 6522 STA DDRA

La partie PIO

La partie PIO fournit deux ports bidirectionnels 8 bits. Chaque port est muni d'un registre d'entrée-sortie, désigné par ORA et ORB, respectivement pour le port A et le port B (voir fig. 2-14). Chaque registre d'E/S est associé à un registre direction, respectivement DDRA et DDRB. Lorsque le bit correspondant du registre direction est mis à 1, la ligne connectée à OR est une *sortie*. C'est une entrée, quand le bit direction est à 0. La polarité a été choisie de telle sorte que toutes les lignes soient en entrée lorsqu'on fait un « reset ».

Ce PIO présente une dissymétrie. Le port A dispose de deux registres d'E/S : l'un avec, et l'autre sans poignée de main.

Utilisation du PIO

Avant d'utiliser le PIO en entrée ou en sortie, il faut charger les registres direction avec la valeur convenable, pour configurer les bits correspondants des registres d'E/S en entrée ou en sortie. Configurons, par exemple, tout le port A en entrée et tout le port B en sortie.

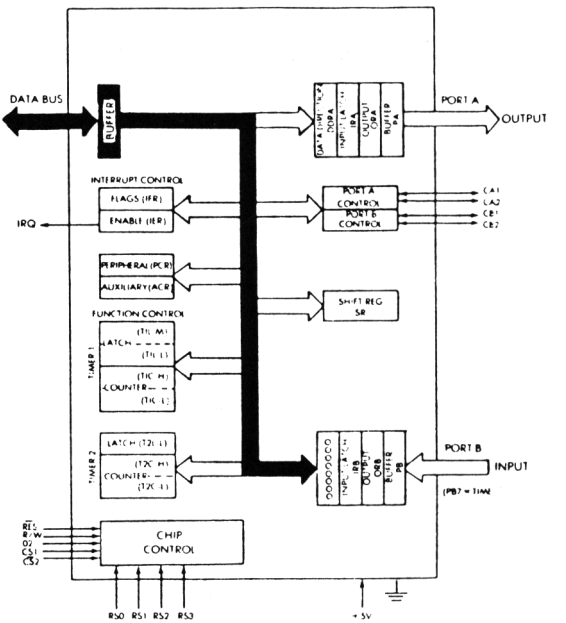


Fig. 2-18 : Utilisation du 6522 : STA DDRB

```
LDA    #$FF        "11111111" = SORTIE
STA    DDRA
LDA    #0
STA    DDRB        B EST EN ENTRÉE
```

(cf. fig. 2-17 et 2-18)

Envoyons maintenant la valeur "00000001" sur le port A (cf. fig. 2-19)

```
LDA    #$01        "00000001"
STA    ORA
```

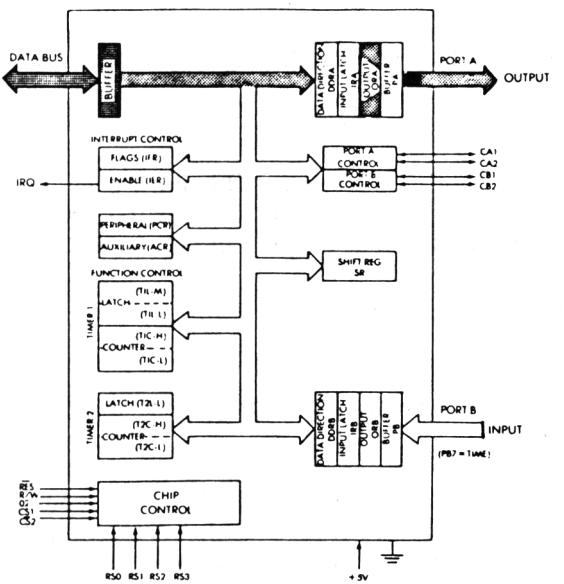


Fig. 2-19 : Utilisation du 6522 : STA ORA

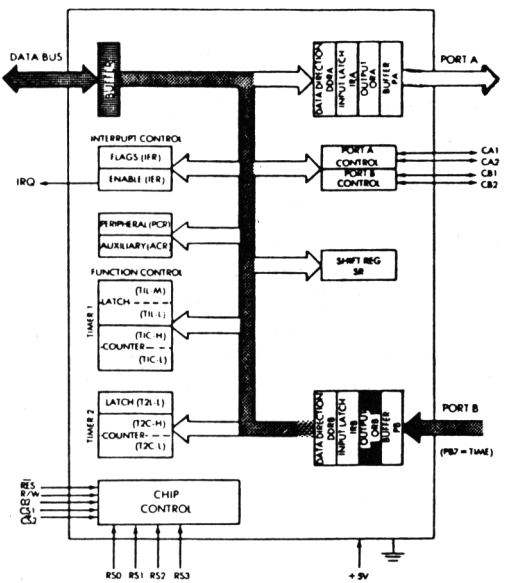


Fig. 2-20 : Utilisation du 6522 : LDA ORB

Lisons la valeur du port B dans l'accumulateur (cf. fig. 2-20) :

```
LDA   ORB
```

Lorsqu'on utilise les registres OR, il est généralement nécessaire de tester un signal *d'état,* pour s'assurer que le périphérique concerné est prêt à recevoir ou à émettre. Cela s'appelle la *poignée de main.* Nous expliquons maintenant le fonctionnement des signaux de commande nécessaires.

Les deux signaux de commande (registre de commande périphérique)

Chaque port est muni de deux lignes de commande, appelées CA1, CA2 et CB1, CB2 (cf. fig. 2-14, à droite). Avant d'envoyer des données à une imprimante, un télétype, par exemple, le microprocesseur doit s'assurer que l'imprimante n'est pas occupée, et qu'elle est prête à accepter le prochain caractère. L'opération est réalisée par une procédure de *poignée de main* (synchronisation des échanges).

Lorsque tel est bien le cas, l'imprimante va envoyer une impulsion ou un front au 6522. Cette transition ou cette impulsion doit être détectée et mémorisée par le 6522, puis décelée par le programme. Le signal sera transmis à l'une des deux entrées de commande CA1 ou CB1.

Le 6522 offre une grande souplesse dans la spécification de la nature du signal reçu ou émis.

Il est possible de préciser si l'on a affaire à une transition de haut en bas (négative = front descendant), ou de bas en haut (positive = front montant), qui déclenchera une interruption interne. Cela est déterminé par le bit 0 (pour CA1) et par le bit 4 (pour CB1) du *registre de commande périphérique* (PCR). "0" correspond à la transition négative et "1" à la transition positive (cf. fig. 2-21).

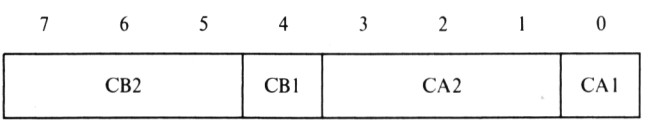


Fig. 2-21 : Registre de commande périphérique (PCR)

7	6	5	4	3	2	1	0
IRQ (lecture présente)	T1	T2	CB1	CB2	SR	CA1	CA2

Fig. 2-22 : Registre indicateur des interruptions (IFR)

7	6	5	4	3	2	1	0
ACTI-VA-TION (Si 1)	T1	T2	CB1	CB2	SR	CA1	CA2

Fig. 2-23 : Registre activation des interruptions (IER)

Une fois la nature du signal spécifiée, il devient possible de le tester.

Test de l'état : Pour détecter si une transition a eu lieu, on peut tester le contenu des bits 1 ou 4 (pour CA1 et CB1 respectivement) du registre indicateur d'interruption (IFR). Le bit reste à 0, tant qu'aucun signal n'a été reçu. Il passe à 1, lorsque la transition appropriée est détectée. Après la lecture d'un état 1, il faut pouvoir remettre le bit à 0, afin de passer à la détection du prochain événement. Deux solutions : 1° écrire un "1" dans la position appropriée du registre, dans le registre d'E/S correspondant.

PCR3	PCR2	PCR1	MODE
0	0	0	Interruption sur front négatif sur CA2. Positionne l'indicateur d'interruption CA2 (IFR0) sur front négatif entré sur CA2. Remet IFR0 à 0, soit par lecture/écriture sur ORA, soit en écrivant 1 sur IFR0.
0	0	1	Interruption sur front négatif sur CA2. Positionne l'indicateur d'interruption CA2 (IFR0) sur front négatif entré sur CA2. La lecture ou l'écriture ne remet pas l'indicateur IFR0 à 0. Seule l'écriture d'un 1 sur le bit IFR0 le fait.
0	1	0	Interruption sur front positif sur CA2. Positionne l'indicateur d'interruption CA2 (IFR0) sur front positif entré sur CA2. Remet IFR0 à 0 soit par lecture/écriture sur ORA soit en écrivant 1 sur IFR0.
0	1	1	Interruption sur front positif sur CA2. Positionne l'indicateur d'interruption CA2 (IFR0) sur front positif entré sur CA2. Seule l'écriture d'un 1 sur IFR0 (pas l'accès à ORA) remet IFR0 à 0.
1	0	0	Sortie poignée de main sur CA2. Met la sortie CA2 à 0 lors d'une lecture ou d'une écriture sur le registre d'E/S ORA. Remet CA2 à 1, lors de la prochaine transition active sur CA1.
1	0	1	Sortie impulsion sur CA2. CA2 passe à 0 pour un cycle après une lecture ou écriture sur le registre d'E/S ORA.
1	1	0	Sortie CA2 manuelle : CA2 est maintenu à 0.
1	1	1	Sortie CA2 manuelle : CA2 est maintenu à 1.

Fig. 2-24 : Fonctionnement détaillé de PCR (courtoisie : Rockwell)

PCR7	PCR6	PCR5	MODE
0	0	0	Interruption sur front négatif sur CB2. Positionne l'indicateur d'interruption CB2 (IFR3) sur front négatif entré sur CB2. Remet IFR3 à 0 soit par lecture/écriture sur ORB, soit en écrivant un 1 sur IFR3.
0	0	1	Interruption sur front négatif sur CB2. Positionne l'indicateur d'interruption CB2 (IFR3) sur front négatif entré sur CB2. Seule une écriture de 1 sur IFR3 remet l'indicateur à 0 (pas une lecture/écriture sur ORB).

0	1	0	Interruption sur front positif sur CB2. Positionne l'indicateur d'interruption CB2 (IFR3) sur front positif entré sur CB2. Remet IFR3 à 0 soit par lecture/écriture sur ORB, soit en écrivant un 1 sur IFR3.
0	1	1	Interruption sur front positif sur CB2. Positionne l'indicateur d'interruption CB2 (IFR3) sur front positif entré sur CB2. Seule, une écriture de 1 sur IFR3 remet l'indicateur à 0 (pas une lecture/écriture sur ORB).
1	0	0	Sortie poignée de main sur CB2. Met CB2 à 0 lors d'une opération d'écriture de ORB. Remet CB2 à 1 lors de la prochaine transition active de l'entrée CB1.
1	0	1	Sortie impulsion sur CB2. Met CB2 à 0 pour 1 cycle après une écriture sur ORB.
1	1	0	Sortie CB2 manuelle : CB2 est maintenu à 0.
1	1	1	Sortie manuelle : CB2 est maintenu à 1.

Fig. 2-25 : Fonctionnement détaillé de PCR

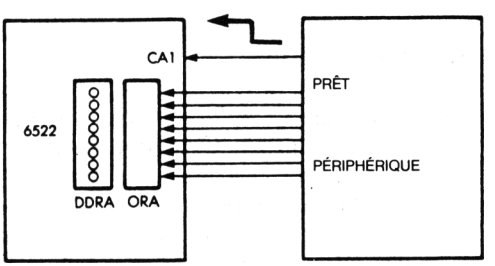


Fig. 2-26 : Lecture de données lorsque le périphérique est prêt

Un exemple simple d'entrée

Spécifions qu'une transition positive signifie « périphérique prêt », et mettons le port A en entrée (cf. fig. 2-26). Lorsque la donnée sera prête, elle sera lue dans l'accumulateur. Voici le programme :

```
LDA   #0
STA   DDRA          MISE EN ENTRÉE
```

```
        LDA   #1
        STA   PCR        TRANSITION POSITIVE
                         ACTIVE SUR CA1
ATTEND  LDA   IFR        LECTURE DES INDICA-
                         TEURS
        AND   #$02       MASQUE LE BIT 1 POUR
                         CA1
        BEQ   ATTEND     PRET ?
        LDA   ORA        LECTURE DES DONNÉES
```

Amélioration : Pouvez-vous modifier les deux instructions "LDA IFR et AND #$02" pour améliorer l'efficacité ?

Fig. 2-27 : Registre de commande auxiliaire du 6522

Verrouillage des entrées-sorties

Les entrées et les sorties du 6522 ne sont pas symétriques. Les sorties sont toujours verrouillées. C'est pourquoi le registre d'E/S est appelé OR (registre de sortie). Les entrées ne sont pas nécessairement verrouillées. Les bits 0 et 1 (respectivement pour le port A et le port B) du registre de commande auxiliaire (ACR) en décident. Lorsque ces bits sont à 0, il n'y a pas de verrouillage en entrée. L'inverse a lieu lorsqu'ils sont à 1 (cf. fig. 2-27). Dans le premier cas le programme lit la valeur actuelle des lignes connectées au port considéré. Dans le second, le verrou est déclenché par la transition active de CA1 ou CB1, selon le port considéré. La valeur est alors conservée dans le registre-tampon jusqu'à la prochaine transition active reçue sur la ligne de commande. Danger : en sortie, le programme lit le registre-tampon qui peut-être différent du contenu de OR.

Envoi d'un signal de commande

CA2 ou CB2 servent à fournir une commande d'échantillonnage (cf. fig. 2-14). Comme ces lignes sont bidirectionnelles, elles doivent

être mises en sortie, en établissant les bits du registre de commande périphérique 3 (pour CA2) ou 7 (pour CB2) (cf. fig. 2-24 et 2-25).

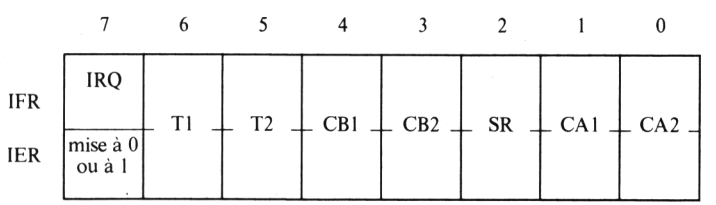
La nature du signal peut être déterminée. Elle sera soit un niveau, soit une impulsion. "0", dans les bits 2 ou 6 (respectivement pour A ou B) correspond à une impulsion. "1" correspond à un niveau. Lorsqu'on spécifie un niveau, on peut aussi spécifier la valeur (haute ou basse) en mettant à 1 ou à 0 les bits 1 et 5 (respectivement pour CA2 et CB2).

Enfin, il est possible de commander la durée d'une impulsion avec les bits 1 et 5, respectivement pour CA2 et CB2. Lorsque le bit est à 0, on aura une impulsion sur un seul cycle. Elle restera à 0 lorsque le bit est à 1, depuis le moment où on accède au registre OR (en lecture ou en écriture) jusqu'à la prochaine transition sur CA1 ou CB1.

Récapitulation de la sortie de commande

On peut spécifier une impulsion, quelles que soient, pratiquement, sa durée et sa polarité. Elle peut être utilisée pour scruter un périphérique (l'interroger), pour acquitter un transfert de donnée, pour passer à un autre périphérique connecté à la même ligne, ou pour commander l'état du périphérique (marche, arrêt, ou toute autre option).

On trouvera un résumé des bits du registre de commande périphérique à la figure 2-21. Les détails sont donnés figures 2-24 et 2-25.



Fg. 2-28 : Registres d'interruptions

Interruptions

Les interruptions sont contrôlées par deux registres : le registre d'activation des interruptions (IER) et le registre des indicateurs d'interruption (IFR) (cf. fig. 2-22, 2-23 et 2-28).

Le registre IFR est le plus souvent utilisé en lecture. Chaque position de 0 à 6 sera à 1, lorsqu'une interruption sera détectée sur les lignes externes (CA1, CA2, CB1, CB2), sur le registre à décalage (SR), ou sur l'un ou l'autre des deux temporisateurs T1 ou T2. Le bit 7 est à 1, dès que l'un des autres bits du registre est à 1.

Le registre d'activation des interruptions (IER) autorise ou inhibe les interruptions dues aux différentes causes. Les bits de IER correspondent à ceux de IFR (cf. fig. 2-28). Lorsqu'un bit est 0, l'interruption correspondante est inhibée. Elle ne sera pas envoyée sur IRQ. Lorsque le bit est à 1, l'interruption est activée. Elle sera enregistrée, si elle se produit. Il est alors possible au programme de lire IFR, et de tester les bits qui l'intéressent, pour déterminer si une interruption a eu lieu. Pour mettre commodément à 1 ou à 0 un bit de IER, on utilise son bit 7 en conjonction avec l'écriture d'un certain motif binaire. Si IER 7 est 0, chaque 1 du motif va mettre à 0 le bit correspondant de IER ; si IER 7 est à 1, chaque 1 va autoriser une interruption.

Exemple : Autorisons les interruptions de CA1 et CA2, et interdisons toutes les autres (se reporter à la fig. 2-28) :

```
LDA   #$7C      "01111100" = METTRE A 0
                LES BITS 2 A 6
STA   IER
LDA   #$83      "10000011" = METTRE A 1
                LES BITS 0 ET 1
STA   IER
```

Exercice 2-1 : *Ecrire un programme qui active l'interruption de CB1 et inhibe les autres.*

Exercice 2-2 : *Inhiber les interruptions de CB1 et CB2, en laissant les autres inchangées.*

Identification de l'interruption

Si plusieurs interruptions se produisent en même temps, autrement dit si plusieurs bits de l'IER sont à 1, le programme devra tester le contenu de IFR, et déterminer quelle interruption s'est produite. L'ordre dans lequel on teste ces bits détermine la priorité des interruptions correspondantes. Par exemple, si

l'interruption due à T1 a la plus haute priorité, le bit correspondant devra alors être testé en premier. La manière la plus simple de tester le contenu de IFR est de le décaler, à droite ou à gauche, d'un cran, et de tester le bit qui tombe dans l'indicateur de retenue. Cette technique attribue les priorités, dans l'ordre, de gauche à droite ou de droite à gauche, aux signaux de la figure 2-28.

Exercice 2-3 : *Regardez la figure 2-28. Listez les dispositifs par ordre de priorité, en supposant que le programme de scrutation décale IFR sur la gauche.*

Naturellement, il est aussi possible de tester des combinaisons d'interruptions en testant des bits spécifiques du registre IFR. Pour plus de détails sur les interruptions et la scrutation, on se reportera au chapitre VI de la référence C3.

Les temporisateurs

Le 6522 possède deux générateurs d'intervalles de temps : T1 et T2, qui peuvent être utilisés en entrée ou en sortie.

Utilisé en sortie, un temporisateur peut générer un signal de sortie ou un train d'impulsions. Utilisé en entrée, il mesure la durée d'une impulsion ou bien compte le nombre d'impulsions reçues. Dans le cas où il génère ou mesure une impulsion de durée donnée, ont dit que le temporisateur fonctionne en mode « monostable ». T1 et T2 du 6522 peuvent être utilisés de cette manière.

Si on utilise le temporisateur pour générer ou compter un train d'impulsions continues, on dit qu'il fonctionne en mode « oscillateur ». Seul T1 peut être utilisé de cette manière.

Avant d'utiliser un temporisateur en mode sortie, il est nécessaire de charger une valeur dans son registre de comptage. En génération d'impulsions, le compteur contiendra, soit le nombre d'impulsions à générer, soit la durée de l'impulsion.

Si on utilise le temporisateur en entrée, son registre doit être mis à 0. En comptage d'impulsions, il contiendra le nombre d'impulsions reçues. En détection d'impulsion, il contiendra la durée de l'impulsion.

T1 comparé à T2

T2 peut être utilisé en entrée, pour compter des impulsions appliquées à la broche PB6 du registre d'E/S du port B

(cf. fig. 2-14). Utilisé en sortie, il ne peut que générer, sur PB6, une impulsion de durée donnée, et non pas générer des trains d'impulsions. Chaque mode est sélectionné à l'aide du bit 5 du registre de commande auxiliaire (ACR) (cf. fig. 2-27). "0" correspond au mode monostable et "1" au mode comptage d'impulsions.

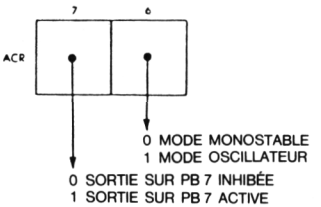


Fig. 2-23 : 6522 : contrôle de T1 par le registre de commande auxiliaire

T1 est différent de T2, et offre des possibilités supplémentaires. Il dispose de quatre modes de fonctionnement (cf. fig. 2-29) et peut être utilisé indifféremment en mode monostable, ou oscillateur. En outre, il peut avoir une sortie sur PB7 activée ou inhibée.

Le mode est déterminé par le bit 6 du registre de commande auxiliaire. Ce bit est "0" pour le fonctionnement en monostable, et "1" en mode oscillateur.

Le bit 7 détermine si PB7 est autorisée ou inhibée. Lorsqu'il est à "0", PB7 est inhibée, lorsqu'il est à 1, elle est activée (fig. 2-30).

ACR7 ACTIVATION SORTIE PB7	ACR6 MODE OSCILLATEUR	MODE
0	0 (MONOSTABLE)	Génère une interruption de time-out lorsque T1 est chargé. PB7 inactive.
0	1 (OSCILLATEUR)	Génère des interruptions continuelles. PB7 inactive.
1	0 (MONOSTABLE)	Génère une interruption et une impulsion sur PB7 à chaque chargement de T1 = monostable et impulsion de largeur programmable.
1	1 (OSCILLATEUR)	Génère des interruptions continuelles et un signal carré sur PB7.

Fig. 2-30 : Modes de fonctionnement du temporisateur 1 du 6522 sélectionnés par le registre de commande auxiliaire

Chargement des compteurs

Chaque temporisateur utilise un compteur sur 16 bits. La partie basse doit être chargée en premier, puis la partie haute. Le chargement de la partie haute du compteur met automatiquement à 0 l'indicateur d'interruption du temporisateur, et démarre le décomptage. T1 est également muni d'un véritable tampon 16 bits, alors que T2 n'en a pas. Ce qui permet au premier de fonctionner continuellement, en mode « oscillateur ». Le tampon est transféré automatiquement dans le compteur quand celui-ci atteint 0. On peut lire et écrire la valeur des tampons de T1 sans affecter les compteurs. Ce procédé est utilisé pour générer toutes les formes d'ondes, aussi complexes soient-elles.

L'adressage des temporisateurs est exposé en détail figure 2-31.

	ADRESSE	ÉCRITURE	LECTURE
TEMPO-RISATEUR 1	– – 04	T1 L-L	T1C-L + RAZ indicateur int. T1
	– – 05	T1L-H + T1C-H + T1L-L→T1C-L + RAZ indicateur T1	T1C-H
	– – 06	T1L-L	T1L-L
	– – 07	T1L-H + RAZ indicateur T1	T1L-H
TEMPO-RISATEUR 2	– – 08	T2L-L	T2C-L + RAZ indicateur int. T2
	– – 09	T2C-H + T2L-L→T2C-L + RAZ indicat. T2	T2C-H

Fig. 2-31 : Adressage des temporisateurs du 6522

Durée exacte

La forme d'onde exacte du temporisateur 1 est montrée par la figure 2-32. Noter que la durée réelle est la valeur du compte ("N") plus 2 ou plus 1,5. Pour obtenir un délai plus exact, l'utilisateur doit donc charger le registre avec le nombre de périodes désiré, moins 2.

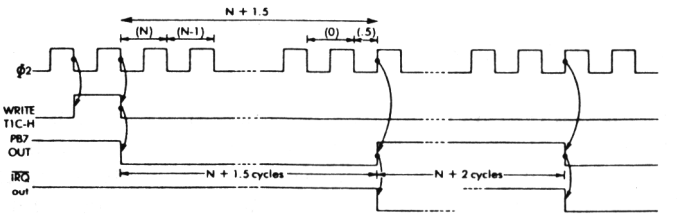


Fig. 2-32 : T1 en mode oscillateur

Le registre à décalage

Le registre à décalage a été inclus pour rendre possible la conversion série-parallèle et parallèle-série. La vitesse de décalage peut être contrôlée par trois sources : le temporisateur T2, la phase 2 de l'horloge (Φ 2) et une horloge externe. La source de synchronisation est spécifiée par les bits 2 et 3 du registre ACR (cf. fig. 2-27). Le bit 4 du registre ACR spécifie le mode entrée ou sortie. La table démontrant la fonction de ces bits apparaît figure 2-33.

ACR4	ACR3	ACR2	MODE
0	0	0	Registre à décalage inactif.
0	0	1	Décalage en entrée sous contrôle de T2.
0	1	0	Décalage en entrée sous contrôle de φ 2.
0	1	1	Décalage en entrée sous contrôle d'impulsions extérieures.
1	0	0	Sortie en mode continu, fréquence déterminée par T2.
1	0	1	Décalage en sortie sous contrôle de T2.
1	1	0	Décalage en sortie sous contrôle de φ2.
1	1	1	Décalage en sortie sous contrôle d'impulsions extérieures.

Fig. 2-33 : Commande du registre à décalage

En sortie, l'utilisateur charge le registre à décalage, et démarre ainsi, automatiquement, le processus de temporisation et de décalage. Lorsque 8 bits auront été décalés hors du registre, l'indicateur d'interruption (bit 2 de IFR) sera positionné. Il pourra alors être testé par le programme.

En entrée, le registre à décalage doit être initialisé à une valeur quelconque, "0" par exemple, pour démarrer le processus. Il commence alors à saisir des bits, à la fréquence de la source de synchronisation spécifiée par les bits 2, 3 et 4 de ACR : T2 ou l'horloge $\varphi 2$ ou une horloge externe. Lorsque 8 bits ont été emmagasinés, l'indicateur d'interruption correspondant dans IFR est positionné. Le programme va déposer une valeur, telle que "0", dans SR, puis tester continuellement la valeur du bit 2 de IFR. A la première interruption, le décalage est terminé. Il est nécessaire d'inhiber le registre à décalage en mettant à 0 les bits 2, 3 et 4 de ACR, tout en stockant les données. Naturellement, si ces dernières viennent en mode continu, le registre à décalage ne sera pas inhibé, et le programme devra « revenir » assez vite pour qu'il n'y ait pas de données perdues.

PROGRAMMATION DU 6522

Le 6522 est une combinaison de PIO, temporisateur et registre à décalage. Les opérations d'entrées-sorties élémentaires sur le PIO sont effectuées pour l'essentiel comme sur le 6520, à ceci près que les registres sont directement accessibles et qu'il n'est pas utile de modifier le bit 2 d'un registre de commande pour les distinguer. On obtient de la sorte une programmation plus simple et plus concise. Néanmoins, les possibilités de commande du 6522 sont très étendues, et très différentes de celles du 6520. Nous examinerons quelques exemples d'entrées-sorties élémentaires avant d'envisager plusieurs cas d'options de commande.

Entrée élémentaire

Une entrée s'accomplit en chargeant des zéros dans le registre direction du port, qui doit se comporter en entrée, puis en lisant le contenu du registre d'E/S OR. Dans le programme simple ci-dessous nous rangerons, en outre, les données qui viennent d'être lues dans la case mémoire 20 :

```
ENTRÉE  LDA  #0
        STA  DDRA     PORT A EN ENTRÉE
        LDA  ORA      LECTURE DES DONNÉES
        STA  $20      SAUVEGARDE
                      EN MÉMOIRE
```

RS3	RS2	RS1	RS0	LECT./ECR.	REGISTRE	COMMENTAIRE
0	0	0	0	E	ORB	
0	0	0	0	L	IRB	
0	0	0	1	E	ORA	Avec poignée de main
0	0	0	1	L	IRA	
0	0	1	0		DDRB	
0	0	1	1		DDRA	
0	1	0	0	E	T1L-L	Tampon
0	1	0	0	L	T1C-L	Compteur
0	1	0	1		T1C-H	Déclenche T1L-L→T1C-L
0	1	1	0		T1L-L	
0	1	1	1		T1L-H	
1	0	0	0	E	T2L-L	Tampon.
1	0	0	0	L	T2C-L	Compteur.
1	0	0	1		T2C-H	Déclenche T2L-L→T2C-L.
1	0	1	0		SR	
1	0	1	1		ACR	
1	1	0	0		PCR	
1	1	0	1		IFR	
1	1	1	0		IER	
1	1	1	1		ORA	Sans poignée de main.

Fig. 2-34 : Les registres du 6522 sont sélectionnés directement

Sortie élémentaire

Une sortie s'effectue exactement comme une entrée. On charge le registre direction du port B avec des uns, exclusivement. Toutes les lignes sont mises en sortie. Les données à envoyer sur le port sont supposées résider dans la case mémoire 20, de sorte qu'elles seront d'abord chargées dans l'accumulateur, puis envoyées sur ORB. On se souvient que le 6502 ne possède pas d'instruction permettant le transfert direct de la mémoire 20 à ORB. Il est donc nécessaire d'avoir recours à une instruction supplémentaire pour, dans un premier temps, transférer de la mémoire vers l'accumulateur, puis de l'accumulateur vers ORB. Le programme apparaît ci-dessous :

```
SORTIE   LDA   #$FF
         STA   DDRB      B EN SORTIE
         LDA   $20       PRISE DES DONNÉES EN
                         MÉMOIRE
         STA   ORB       SORTIE SUR ORB
```

Utilisation des options de commande

Nous configurerons ici le port A tout entier en entrée, et supposerons que le périphérique connecté au port A envoie le signal « donnée prête » sur la ligne CA1. La transition positive sera active. Le 6522 devra détecter cette transition, et le programme le scrutera pour déterminer si une donnée a été reçue. Si tel est bien le cas, il lira la donnée et la rangera en mémoire à l'adresse 20. Le programme, déjà donné partiellement (cf. « entrée élémentaire » p. 47), apparaît ci-dessous :

```
PRÊT    LDA   #0        A = ENTRÉE
        STA   DDRA
        LDA   #1        INT CA1 POSITIVE
        STA   PCR
TEST    LDA   IFR       TEST DU BIT 1
        AND   #$2       00000010 EN BINAIRE
        BEQ   TEST       = 1 ?
        LDA   ORA       LECTURE DE LA DONNÉE
        STA   $20       SAUVEGARDE
                        EN MÉMOIRE
```

Comme d'habitude, le registre direction est mis à 0 pour configurer tous les bits de ORA en entrées :

```
LDA   #0
STA   DDRA
```

Le registre de commande PCR va maintenant être configuré, pour qu'une interruption interne puisse être générée lorsqu'il arrive une transition positive sur CA1 :

```
LDA   #1
STA   PCR
```

Ces deux instructions chargent la valeur 0000001 dans PCR. En se reportant à la figure 2-21, le lecteur vérifiera qu'il s'agit bien de la valeur correcte. Le bit 0 du registre de commande PCR spécifie quelle transition du signal d'entrée sera reconnue. Comme nous voulons que l'indicateur soit positionné sur transition positive (de 0 à 1) de CA1, PCR0 doit être mis à 1.

Les bits 6 et 7 de ACR concernent le temporisateur T1. Leur valeur est indifférente, puisque T1 n'est pas utilisé ici. Les bits 2, 3 et 4 de ACR déterminent le fonctionnement du registre à décalage. Ils doivent être à 0, comme le montre la figure 2-33, dans la mesure

où le registre n'est, lui non plus, pas utilisé. Le bit 5 de ACR concerne T2, et ne sera pas pris en considération. Le bit 1 met en action le verrouillage sur PB. Le bit 0 active le verrouillage sur le port A. Lorsqu'on le demande (en écrivant un "1"), les données présentes sur les entrées A sont mémorisées lors de la transition active sur CA1. On aurait ainsi :

```
LDA   #1
STA   ACR
```

Nous sommes supposés ici procéder par scrutation, et non par interruptions. Le programme aura donc la responsabilité de lire le contenu du registre indicateur IFR, et de déterminer si un événement a eu lieu. Le contenu du registre IFR apparaît figure 2-28. La position 1 de IFR doit être testée pour savoir si le signal « donnée prête » a été reçu sur CA1.

```
TEST    LDA   IFR
        AND   #$02
        BEQ   TEST
```

L'instruction AND masque tous les bits, à l'exception du 1, qui doit être testé.

Tant que le bit 1 est à 0, le programme reste dans la boucle de test. Une fois le signal « donnée prête » reconnu, les données peuvent être lues sur ORA, et transférées en mémoire à l'adresse 20, comme nous le supposons d'habitude :

```
LDA   ORA
STA   $20
```

La lecture de ORA dans l'accumulateur va automatiquement remettre à 0 le bit 1 de IFR (l'indicateur d'interruption de CA1), de sorte que la condition d'interruption interne sera automatiquement annulée.

Il est important de se rappeler que les indicateurs d'interruption doivent explicitement être remis à 0, à chaque fois qu'on les utilise. Le 6522 est organisé de telle façon que l'opération « normale » qui suit la détection d'une interruption (1), s'en charge automatiquement. Néanmoins, en cas de programmation « non standard », il peut se produire que, par erreur, l'indicateur d'interruption risque d'être à 1 continuellement. Une technique pourrait être utilisée dans un tel cas. Elle consisterait à réécrire IFR après l'avoir lu :

(1) Par exemple, la lecture de ORA, après l'interruption due à CA1.

```
        STA   IFR
```

Cette astuce de programmation ne remet à 0 que le bit auparavant mis à 1. Elle annule ce bit sans modifier les autres (sauf si plusieurs bits étaient à 1).

Protocole de poignée de main

Nous supposerons ici qu'on utilise une séquence de poignée de main complète. Le programme a d'abord la responsabilité d'envoyer une impulsion de départ (active haute) au périphérique. Ce dernier répond par un signal « donnée prête » (actif bas ici). Le programme devra détecter si ce signal a été reçu, puis transférer les données dans la case mémoire 20. Le programme apparaît comme suit :

```
PMAIN   LDA   #0
        STA   DDRA      A ENTRÉE
        STA   ACR
        LDA   #$0C      BITS 2 ET 3 A 1
        STA   PCR       RAZ SIGNAL DÉPART
        LDA   #$0E      BITS 1, 2, 3 A 1
        STA   PCR       GÉNÈRE DÉPART
                        SUR CA2
        LDA   #$0C
        STA   PCR       LE REMET A 0
ATTEND  LDA   IFR
        AND   #$02      INTERRUPTION CA1 ?
        BEQ   ATTEND    BOUCLE DE SCRUTATION
        LDA   ORA       DONNÉE PRÊTE
        STA   $20       SAUVEGARDE
                        EN MÉMOIRE
```

Examinons le programme. Comme d'habitude, le port A est mis en entrée en envoyant des zéros dans DDRA :

```
        LDA   #0
        STA   DDRA      MISE A 0 DE DDRA
        STA   ACR
```

On fera ici l'économie d'un verrouillage sur l'entrée (1). Le registre PCR doit maintenant générer une impulsion de départ,

(1) Cf. Programme précédent, si vous souhaitez verrouiller les données en entrée.

active haute. Le niveau de CA2 (1) sera d'abord mis à 0, puis à 1, pour garantir un front positif. La mise à 0 de la sortie CA2 s'accomplit en donnant la valeur "110" respectivement aux bits 3, 2 et 1 de PCR (cf. fig. 2-24), grâce aux instructions suivantes :

```
LDA   #$0C        00001100
STA   PCR
```

On doit, ensuite, mettre à 1, le niveau de la sortie CA2, en chargeant la valeur "111" dans les bits 3, 2, 1 de PCR :

```
LDA   #$0E        00001110
STA   PCR
```

On admettra qu'une brève impulsion est suffisante pour fournir le signal de « départ », bien que certains périphériques puissent exiger une impulsion d'une certaine durée. Dans un tel cas, il convient de marquer un délai à cet endroit, pour garantir que CA2 restera haut pendant un temps déterminé. Ici, nous remettons simplement le signal à 0 :

```
LDA   #$0C        00001100
STA   PCR
```

A ce point, nous procéderons, comme dans le programme précédent, en scrutant le bit 1 de IFR, pour détecter si CA1 a été mis à 1 :

```
ATTEND  LDA   IFR
        AND   #$02        00000010
        BEQ   ATTEND
```

De la même façon, on lit ensuite la donnée de ORA, et on la range dans la case mémoire 20 :

```
LDA   ORA
STA   $20
```

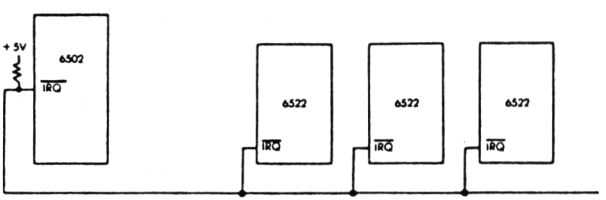


Fig. 2-35 : Connexion de plusieurs 6522 : génération de IRQ

(1) Nous utiliserons cette ligne pour fournir le signal de départ. CA1, qui ne peut être qu'une sortie, est inutilisable.

Utilisation de plusieurs 6522

Dans le cas où on utilise plusieurs 6522, leur sortie de demande d'interruption est habituellement connectée à la ligne IRQ du 6502 (fig. 2-35). Après la réception d'une IRQ par le 6502, le programme doit cependant déterminer de quel 6522 elle émane. On utilise généralement une boucle de scrutation, qui va interroger, tour à tour, l'IFR de chaque boîtier pour repérer lequel d'entre eux a engendré l'interruption. Cette information est disponible, toute prête, dans le bit 7 du registre d'indicateurs d'interruptions, comme le montre la figure 2-22. On sait que le bit 7 est universellement utilisé comme position de test, de préférence à toute autre. Pourquoi ? Parce qu'une fois le contenu du registre à tester chargé dans l'accumulateur, la valeur du bit 7 conditionne le bit de signe du registre d'états du microprocesseur (bit N). La prochaine instruction du programme peut facilement tester le bit « N », et déterminer sa valeur (0 ou 1). C'est exactement ce que fait le programme de scrutation suivant :

```
          LDA   IFR1
          BPL   SUIV1
INTER     ....          (IDENTIFIER LA
                        CAUSE PARMI 7)

SUIV1     LDA   IFR1
          BPL   SUIV2
```

Le programme charge le contenu de l'IFR du premier 6522 dans l'accumulateur, et teste s'il est positif. S'il l'est, cela veut dire qu'aucune interruption n'a été générée par le premier 6522. Le programme procède alors au test suivant, et ainsi de suite. Toutefois, si on découvre qu'un boîtier a généré une interruption, une routine spécifique doit être mise en œuvre pour déterminer la suite des opérations. Examinons-la.

Identification d'une cause d'interruption, parmi 7 possibles, dans le 6522.

En se reportant à la figure 2-22, on peut voir qu'il existe sept conditions susceptibles de déclencher une interruption interne dans le registre IFR du 6522 : T1, T2, CB1, CB2, SR, CA1, CA2. Si l'on utilise toutes les ressources internes du 6522 en même temps, comme c'est souvent le cas, il est nécessaire de tester toutes les possibilités. On trouvera, ci-dessous, un programme simple identifiant une cause d'interruption, parmi 7 possibles :

```
UNDANS7  ASL   A
         BMI   TEMPO1
         ASL   A
         BMI   TEMPO2
         ASL   A
         ....
```

Le programme teste successivement les bits 6, 5, 4, etc., en décalant, à chaque fois, le contenu de l'accumulateur d'un cran à gauche. Il faut noter que l'ordre des décalages établit une priorité parmi les interruptions d'un 6522. Si on utilise le programme ci-dessus, le temporisateur T1 aura la plus grande priorité, puis T2, etc. L'utilisateur peut, s'il le veut, assigner des priorités différentes en testant les bits dans un autre ordre.

Génération d'un délai avec un temporisateur

Il est indispensable d'étudier en détail les particularités des temporisateurs dans les fiches techniques des constructeurs, avant de les utiliser. T2 est plus simple que T1. Les deux temporisateurs ne sont pas identiques. Dans la mesure où une étude complète des temporisateurs n'est pas nécessaire pour la suite de ce livre, nous nous bornerons à présenter deux exemples-type de génération de délais utilisant respectivement T2 et T1. D'autres exemples seront présentés dans les chapitres d'applications.

Génération d'un délai avec T2 en mode monostable

Le programme apparaît ci-dessous :

```
MONOST2  LDA   #0
         STA   ACR          CHOIX DU MODE
         STA   T2LL         TAMPON BAS = 0
         LDA   #$01         DURÉE PARTIE HAUTE
         STA   T2CH         = 01 HEXA DÉPART
         LDA   #$20         MASQUE
BOUCLE   BIT   IFR          DÉLAI ÉPUISÉ
         BEQ   BOUCLE
         LDA   T2CL         RAZ INTER T2
```

Le bit 5 de ACR doit être mis à 0 pour spécifier le mode retenu pour T2 (délai fondé sur φ 2 et non sur l'entrée PB6).

Nous supposons qu'aucune autre ressource, par exemple le registre à décalage, n'est ici utilisée. Tout le registre ACR est donc mis à 0 :

```
LDA   #0
STA   ACR
```

Comme T1, T2 contient un registre de 16 bits, dont les deux moitiés doivent être chargées tour à tour, en commençant par la partie basse.

```
STA   T2LL
LDA   #$01
STA   T2CH
```

Le chargement de la valeur $01 dans T2C-H a automatiquement pour effet de remettre à 0 l'indicateur d'interruption, et de démarrer le comptage.

La figure 2-28 montre que le bit 5 de IFR a pour rôle d'indiquer que T2 est arrivé au bout de son délai. Il faut donc tester si IFR-5 a la valeur 1, à l'aide des trois instructions qui suivent :

```
        LDA   #$20        BIT 5 = 1
BOUCLE  BIT   IFR
        BEQ   BOUCLE
```

La valeur 20 hex est égale à "00100000". Elle sert à tester le bit 5.

L'instruction BIT effectue un ET logique, sans modifier le contenu de l'accumulateur. Tant que le bit 5 reste à 0, le programme boucle, en attente de l'interruption de T2. Quand cette dernière a lieu, elle est détectée, et le programme sort de la boucle.

Le programme doit enfin explicitement remettre à zéro l'indicateur d'interruption T2, avant de passer à une autre tâche. On pourrait, pour ce faire, recharger une nouvelle valeur dans le compteur. Mais comme ce programme doit pouvoir être utilisé quel que soit l'environnement, nous nous garderons de faire aucune hypothèse sur ce qui pourra être fait ultérieurement. L'indicateur d'interruption est remis à 0, soit en écrivant dans T2 C-H, soit en lisant T2 C-L. Puisqu'il n'est pas souhaitable de redémarrer le compteur, nous retiendrons plutôt la deuxième solution pour mettre à 0 l'interruption :

```
LDA   T2CL
```

Génération d'un délai avec T1 en mode monostable

Nous utilisons ici T1 d'une façon analogue à celle mise en œuvre ci-dessus avec T2. T1 est, toutefois, muni d'un véritable tampon 16 bits, contrairement à T2. Le programme apparaît ci-dessous :

```
MONOST1  LDA   #0        MODE MONOSTABLE.
         STA   ACR       PAS D'IMPULSION SUR PB7
         STA   T1LL      TAMPON BAS
         LDA   #$01      DÉLAI
         STA   T1CH      CHARGE AUSSI T1CL
                         ET DÉMARRE
BOUCLE   BIT   IFR       DÉLAI ÉPUISÉ ?
         BVC   BOUCLE
         LDA   T1LL      RAZ
                         INDICATEUR D'INTER
```

Ce programme est, pour l'essentiel, analogue au programme précédent : il devrait se comprendre de lui-même. La seule différence réside dans le fait qu'il est inutile de charger un masque pour le tester, puisque l'indicateur est le bit 6 de IFR. L'instruction BIT le transfère dans l'indicateur d'état V, qu'il suffit de tester par BVC. Le reste du programme est identique.

Génération d'une impulsion

Les programmes précédents génèrent un délai. Pour générer une impulsion proprement dite, il faut spécifier la broche de sortie convenable. Pour T1, la broche PB7 sera utilisée pour fournir l'impulsion en sortie. PB7 sera en sortie : soit si DDRB7, soit si ACR7 égale 1.

T2 n'envoie pas directement une impulsion sur une broche de *sortie*. L'impulsion doit être produite en ajoutant des instructions, qui mettent explicitement à 1, puis à 0, une broche d'un port. En revanche, T2 peut *compter* des impulsions en mode comptage. On utilise, pour cela, la broche PB6. C'est dire que les différences pratiques entre les temporisateurs se trouvent, une nouvelle fois, soulignées. Dans toute application pratique, le lecteur est encouragé à revoir les fiches techniques du constructeur, pour les exploiter au mieux.

Décalage en entrée et en sortie

Le registre à décalage SR est connecté à la broche CB2 du 6522. Toutes les impulsions sont sorties ou lues sur cette broche. La combinaison des bits 2, 3 et 4 de ACR détermine le mode de fonctionnement du registre à décalage. Les 8 combinaisons possibles sont décrites figure 2-33.

Dans les exemples envisagés jusqu'ici, les trois bits ont toujours été 0, de sorte que le registre à décalage était inactif. Le registre décale en entrée ou en sortie, sous contrôle de l'une des trois causes possibles de synchronisation : T2, phase 2 de l'horloge, ou horloge extérieure. En outre, il possède un mode spécial, avec sortie continue, à la fréquence déterminée par T2. Le lecteur se reportera, à nouveau, aux fiches techniques du constructeur, pour obtenir les spécifications complètes. Nous présentons ici, simplement, deux exemples caractéristiques de décalage en entrée et en sortie.

Décalage en entrée sous contrôle d'une horloge extérieure

```
DECENT   LDA   #0
         STA   ACR          RAZ
         LDA   #$0C         MODE HORLOGE EXT
         STA   ACR          DÉPART
BOUCLE   LDA   IFR          INDICATEUR
         AND   #$04         TESTE BIT 2
         BEQ   BOUCLE       BOUCLE D'ATTENTE
         LDA   SR           LIT LES 8 BITS
                            DANS L'ACCU
         STA   $20          SAUVEGARDE
                            EN MÉMOIRE
```

On commence par désactiver le registre à décalage en chargeant des zéros dans ACR :

```
LDA   #0
STA   ACR
```

Ensuite, on spécifie le mode de fonctionnement recherché, en chargeant la valeur "011" dans les bits 4, 3 et 2 de ACR :

```
LDA   #$0C
STA   ACR
```

Un décalage en entrée se trouve ainsi spécifié, sous contrôle d'une horloge extérieure (cf. fig. 2-33).

Lorsque 8 décalages ont eu lieu, le mécanisme est automatiquement stoppé, et l'indicateur d'interruption due à SR est mis à 1, dans IFR. Après le début du décalage, le programme a donc simplement à charge de tester le bit 2 de IFR (cf. fig. 2-28), pour vérifier qu'il est à 1. La boucle d'interrogation apparaît ci-dessous :

```
BOUCLE   LDA   IFR
         AND   #$04
         BEQ   BOUCLE
```

A ce point, il est simplement nécessaire de transférer le contenu de SR en mémoire, à l'adresse 20, comme d'habitude :

```
         LDA   SR
         STA   $20
```

Décalage en sortie sous contrôle de la phase 2

Le programme est, pour l'essentiel, semblable au précédent, à ceci près que les bits de commande à charger dans ACR sont différents, afin de spécifier le mode de fonctionnement voulu. Dans l'hypothèse où on n'envoie à l'extérieur qu'un seul mot de 8 bits, il n'est nul besoin de boucle d'attente, pour déterminer la fin éventuelle du décalage. Le programme apparaît ci-dessous :

```
DECSOR   LDA   #0
         STA   ACR          RAZ
         LDA   #$18
         STA   ACR          MODE SORTIE/Φ 2
         LDA   $20          LECTURE DES DONNÉES
                            EN MÉMOIRE
         STA   SR
```

Comme précédemment, on commence par remettre ACR à 0 avant d'y charger la valeur "18", en hexadécimal, qui correspond à la combinaison "110" dans les positions 4, 3 et 2. Cela veut dire : « *décalage en sortie sous contrôle de la phase 2 de l'horloge système* » :

```
         LDA   #0
         STA   ACR
```

```
LDA   #$18
STA   ACR
```

On cherche alors la donnée en mémoire, à l'adresse 20, et on la dépose dans le registre à décalage. L'écriture dans le registre démarre automatiquement le mécanisme de décalage :

```
LDA   $20
STA   SR
```

Si nous avions à envoyer une suite de mots de 8 bits, le programme devrait, dans ce cas, attendre qu'une opération de 8 décalages soit terminée, avant de commencer la suivante. Il faudrait donc avoir recours à une boucle d'attente analogue à celle utilisée ci-dessus. Lorsque 8 bits ont été décalés et sortis, le 6522 a pour caractéristique de mettre automatiquement à 1 le bit 2 de IFR (cf. fig. 2-28). Dans notre hypothèse, le programme testerait simplement IFR2 en continu, jusqu'à ce qu'il passe à 1. Une fois cette valeur détectée, on opérerait un nouveau décalage.

Récapitulation du 6522

Les trois fonctions de ce composant sont : PIO, temporisateur et décalage. Il est, de surcroît, possible de demander des fonctions de commande sophistiquées, pour le PIO et les temporisateurs. Nous avons décrit le fonctionnement des signaux de commande possibles, et de leurs différentes options. Le 6522 doit être considéré comme un ensemble de trois fonctions séparées. Les fonctions des ports A et B sont, pour l'essentiel, similaires, mais non symétriques. Les deux temporisateurs ont des caractéristiques communes, mais offrent des possibilités différentes. Enfin, le registre à décalage dispose de fonctions d'entrée et de sortie essentiellement symétriques. Il peut être utilisé pour émettre ou recevoir des bits ou des mots en série, à une fréquence quelconque, établie par de nombreuses sources extérieures.

Exercice 2-4 : *Sauver dans une table de deux mots-mémoire d'adresse TAMPON, deux mots de données successifs venant du périphérique P1. P1 fournit un signal « PRET » sous forme de front montant. Il nécessite un signal d'acquittement impulsion positive.*

Exercice 2-5 : *Analogue au 2-4, mais P1 exige une impulsion « DEPART » active basse, et répond avec le signal PRET.*

Exercice 2-6 : *Envoyer ces données vers le périphérique P2, à partir de l'adresse-mémoire TAMPON. P2 fournit un signal OCCUPE lorsqu'il n'est pas prêt.*

Exercice 2-7 : *Analogue à 2-6, mais P2 nécessite une impulsion d'échantillonnage ETAT pour fournir une réponse PRET/OCCUPE.*

Exercice 2-8 : *Demarrer une imprimante avec un « 1 » sur la ligne de commande, attendre « PRET », envoyer un caractère, mettre à l'arrêt.*

Exercice 2-9 : *Compter 10 impulsions entrées sur PB6.*

Exercice 2-10 : *Générer une impulsion de 1 ms sur PB7.*

Exercice 2-11 : *Décaler vers l'extérieur 8 bits pris à l'adresse TAMPON au rythme de T2.*

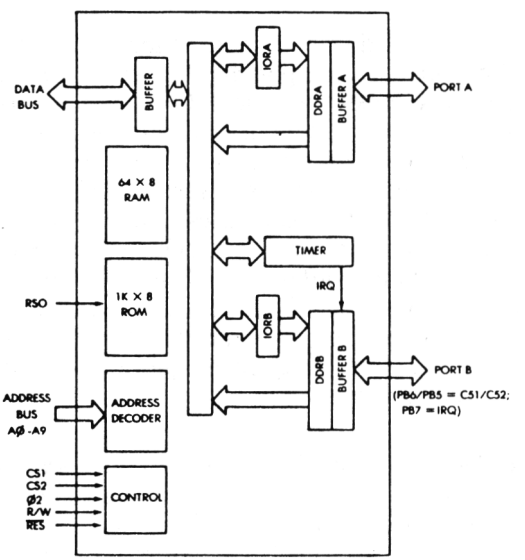


Fig. 2-36 : Architecture interne du 6530

LE 6530 : ROM, RAM, E/S, TEMPORISATEUR
(appelé en abrégé RRIOT : ROM, RAM, I/O, TIMER)

Le 6530 est un composant combiné spécial, qui réunit quatre fonctions habituellement séparées : un PIO, un temporisateur, une RAM et une ROM. Son architecture interne est montrée figure 2-36. Il est muni des deux ports de PIO habituels, pourvus de registres direction propres. En revanche, il ne dispose pas de lignes de commande, ni de logique d'interruption, associées aux ports. Le temporisateur est relié au port B. La mémoire RAM comprend 64 octets, et la ROM 1K octets. Cette dernière, une fois programmée, ne peut pas être changée. Comme il n'est pas rentable de produire des ROM en faibles quantités, le 6530 n'est utilisé que lorsque l'on prévoit de produire un grand nombre de composants identiques. Par exemple, la carte KIM utilise deux 6530, qui contiennent le programme de gestion interne, autrement dit le « moniteur ».

Trois broches de ce composant ont une double fonction : CS1 et CS2 sont des options du masque, en remplacement de PB6 et PB5 ; PB7 pour sa part, peut être aussi utilisée comme demande d'interruption IRQ.

Le temporisateur

Le générateur d'intervalles de temps est muni d'un registre 8 bits. Il peut être utilisé sous quatre modes. Selon la valeur de A0 et A1 du bus adresse, il compte en unités égales à 1, 8, 64 ou 1024 fois la période d'horloge du système. Au programmeur, il apparaît comme un ensemble de 4 emplacements mémoire, comme le montre la figure 2-37.

A2	A1	A0	
0	0	0	TAMPON A
0	0	1	DDRA
0	1	0	TAMPON B
0	1	1	DDRB
1	0	0	TEMPO 1TM
1	0	1	(E) TEMPO 8T (L) INDIC INTER
1	1	0	TEMPO 64T
1	1	1	(E) TEMPO 1024T (L) INDIC INTER

Note : A3 spécifie si l'interruption est utilisée :
A3 = 0 : PAS D'IRQ SUR PB7
A3 = 1 : IRQ SUR PB7

Fig. 2-37 : Carte d'implantation-mémoire du 6530

Lorsqu'on utilise le temporisateur, la broche PB7 peut servir de broche d'interruption. Utilisée comme IRQ, elle doit être programmée comme entrée. Dans tous les autres cas, elle sera utilisée de manière habituelle.

Pour plus de détails sur l'utilisation de PB7 en interruption, le lecteur est invité à se reporter aux fiches techniques du constructeur.

LE 6532
(RIOT = RAM, I/0, TIMER : RAM, E/S, TEMPORISATEUR)

Le 6532 est pour l'essentiel un 6530, sans la ROM. La RAM est, en revanche, plus grande : elle fournit 128 octets. D'autre part, la ligne PA7 du 6532 peut-être utilisée comme entrée détectrice de front. Lorsqu'on a recours à cette possibilité, une transition active positionne un indicateur d'interruption interne (bit 6 du registre d'indicateurs d'interruption).

L'architecture interne du 6532 apparaît figure 2-38, et l'adressage de ce boîtier figure 2-39. Pour le reste, le fonctionnement du 6532 est pratiquement semblable à celui du 6530.

Les ports A et B ne sont pas symétriques. Principale différence : le port B est muni de tampons actifs, capables de fournir 3 mA sous 1,5 V. On peut ainsi connecter directement ce port à des LED, ou à des transistors Darlington. De plus, le port A lit directement l'état des broches, tandis que, sur le port B, c'est le registre de sortie qu'on lit.

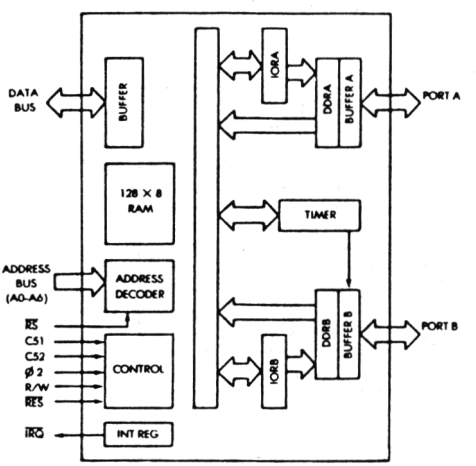


Fig. 2-38 : Architecture interne du 6532

RS	A4	A3	A2	A1	A0	R/W	SÉLECTION
0	–	–	–	–	–	–	RAM
1	–	–	0	0	0	–	ORA
1	–	–	0	0	1	–	DDRA
1	–	–	0	1	0	–	ORB
1	–	–	0	1	1	–	DDRB
1	1	*	1	0	0	0	ÉCRITURE TEMPORISATEUR 1T
1	1	*	1	0	1	0	ÉCRITURE TEMPORISATEUR 8T
1	1	*	1	1	0	0	ÉCRITURE TEMPORISATEUR 64 T
1	1	*	1	1	1	1	ÉCRITURE TEMPORISATEUR 1024T
1	–	*	1	–	0	1	LECTURE TEMPORISATEUR
1	–	–	1	–	1	1	LECTURE INDICATEUR INTERRUPTION
1	0	–	1	**	***	0	ÉCRITURE COMMANDE DÉTECTEUR DE FRONTS

– Indifférent.
* Active (1)/inhibe (0) interruption du temporisateur vers IRQ.
** Active (1)/inhibe (0) interruption de PA7 vers IRQ.
*** Détection de front négatif (0)/positif (1).

Fig. 2-39 : Adressage du 6532

RÉSUMÉ

La plupart des applications nécessitent l'utilisation d'un ou deux ports, sur un ou plusieurs PIO, ainsi que d'un temporisateur programmable. Les plus complexes exigent des signaux de commande et, le cas échéant, des décalages automatiques. Tous les composants que nous avons passés en revue – le 6520, le 6522, le 6530 et le 6532 – possèdent deux ports de PIO. A l'exception du 6520, ils ont tous, au moins, un temporisateur programmable. Un tableau de comparaison de ces quatre composants d'entrée-sortie apparaît figure 2-40.

On utilisera au moins un de ces PIO dans chaque application présentée dans ce livre.

	6520	6522	6530	6532
LIGNES DU PORT A	8	8	8	8
LIGNES DU PORT B	8	8	5 à 8	8
LIGNES COMMANDE A	2	2	0	0
LIGNES COMMANDE B	2	2	0	0
DDRA	OUI	OUI	OUI	OUI
DDRB	OUI	OUI	OUI	OUI
TEMPORISATEUR 1	–	OUI	OUI	OUI
TEMPORISATEUR 2	–	OUI	–	–
ROM	–	–	1K × 8	–
RAM	–	–	64 × 8	128 × 8
AUTRE	–	1 registre comm. suppl.	4 unités de temps	4 unités de temps
INTERRUPTIONS	2	1	sur option	1

Fig. 2-40 : Tableau de comparaison des 4 PIO

3

SYSTÈMES A 6502

INTRODUCTION

Les applications présentées dans ce volume seront connectées à un système 6502 standard, qu'il convient donc d'abord de présenter. Nous décrirons ensuite certaines cartes réelles à 6502, et montrerons qu'elles sont conformes au modèle standard.

Pour présenter des applications réalistes, il est nécessaire de définir exactement la configuration hardware à laquelle les applications sont effectivement connectées. La majorité des exemples présentés sont directement applicables à la carte SYM, et peuvent être immédiatement adaptés à la carte KIM. Une section du chapitre suivant présentera explicitement des programmes pour le KIM. SYBEX ne recommande en particulier aucune carte, ni aucun constructeur. Simplement, dans une perspective pédagogique, il nous est apparu plus pratique de présenter des applications directement implantables sur des cartes existantes, plutôt que d'inventer une carte fictive. La plupart des programmes écrits pour le SYM sont compatibles avec le KIM, et peuvent facilement s'adapter à d'autres cartes, comme l'AIM 65. Nous laissons le lecteur seul juge du choix de la carte la mieux adaptée à ses besoins.

Nous présenterons, dans ce chapitre, les architectures du KIM, du SYM et de l'AIM 65. Le SYM sera présenté plus en détail, pour que le lecteur qui ne possède pas de SYM puisse comprendre les connexions utilisées dans les applications présentées dans les chapitres suivants. Toutefois, nous le répétons, n'importe quelle autre carte peut être utilisée, avec des changements au programme généralement mineurs. De même, si on lui ajoute une extension comprenant un ou deux boîtiers d'entrée-sortie comme le 6522, le P.E.T./C.B.M. peut, lui aussi être utilisé, avec des adaptations très minimes des programmes.

UN SYSTÈME A 6502 « STANDARD »

Tout système standard à microprocesseur comprend, au moins, le microprocesseur (MPU) et son circuit d'horloge, la ROM, la RAM, et un ou plusieurs PIO. L'organisation d'un tel système, basé sur le 6502, est détaillée figure 3-1.

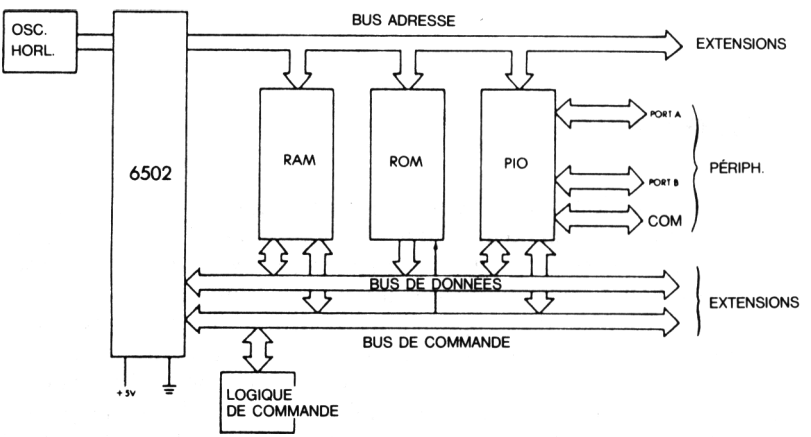


Fig. 3-1 : Organisation d'un système à 6502 standard

La plus grande partie du circuit d'horloge est intégrée dans le boîtier (1) : on n'a besoin, à l'extérieur, que d'un oscillateur et d'un quartz. Comme tout microprocesseur standard, le 6502 crée trois bus : le bus adresse (16 lignes), le bus de données (8 lignes bidirectionnelles) et le bus de commande.

Dans ce type de système, la RAM (mémoire à lecture-écriture), la ROM (mémoire à lecture seule) et le PIO, sont montrés sous la forme de boîtiers séparés, connectés aux 3 bus. La ROM contiendra gnéralement un programme moniteur, nécessaire pour exploiter les ressources de la carte. Dans les applications industrielles, elle contiendra les programmes de l'utilisateur. Le PIO crée deux ports (8 lignes chacun), pour communiquer avec des appareils extérieurs, et même, éventuellement, quelques lignes de commande supplémentaires. Dans toute application pratique, il faudra au moins deux

(1) Voir la partie gauche de la figure.

PIO pour fournir un nombre de lignes d'E/S suffisant. Le décodage d'adresses et d'autres fonctions requiert généralement l'introduction d'un peu de logique supplémentaire.

Plusieurs boîtiers combinés sont disponibles dans la famille 6502. La ROM, la RAM et le PIO peuvent être, en conséquence, combinés sur un ou deux boîtiers. En tous cas, tout système à base de 6502 incorporera normalement tous les éléments logiques de la figure 3-1.

Examinons maintenant quelques cartes réelles, et voyons de quelle manière elles correspondent à notre carte standard.

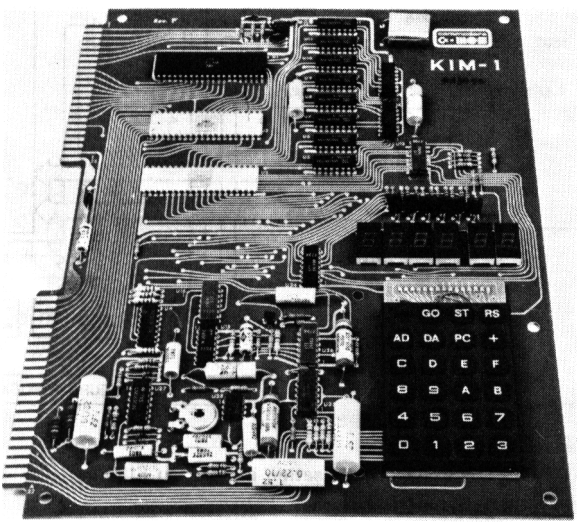


Fig. 3-2 : Photo du Kim-1

LE KIM-1

Le KIM-1 est une carte introduite depuis longtemps par MOS-Technology, pour appuyer leur microprocesseur. Il contient un minimum de composants. Equipé d'un clavier hexadécimal, et de 6 afficheurs à LED, il peut être utilisé comme micro-ordinateur complet, autonome et peu coûteux. La figure 3-3 détaille son organisation interne.

Le KIM-1 possède une RAM séparée de 1 K par 8 (pour l'utilisateur), et deux boîtiers combinés 6530. On sait, depuis le

chapitre précédent, que le 6530 est un boîtier combinant un PIO, un temporisateur programmable, une ROM et une RAM. Il est inutile que cette carte soit pourvue d'une mémoire ROM externe, puisque la quantité de mémoire fournie par les deux 6530 est suffisante pour contenir le système moniteur. Chaque 6530 contient également 64 octets de RAM, qui sont, en partie, utilisés par le moniteur.

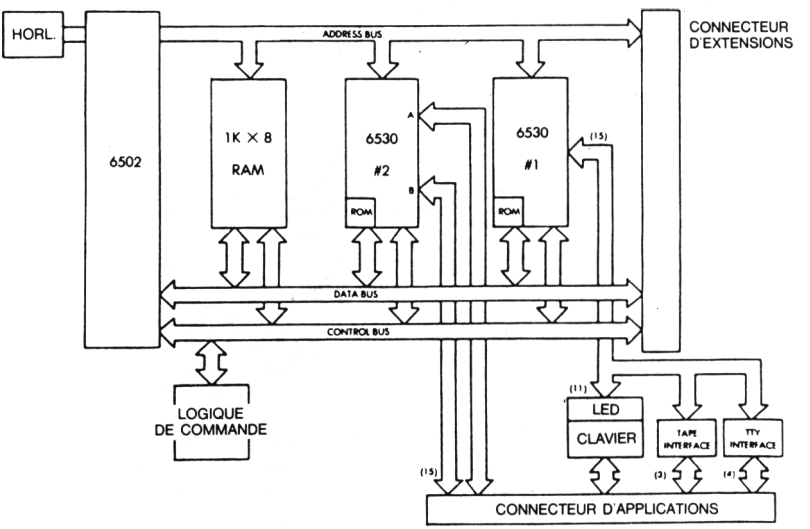


Fig. 3-3 : Organisation interne du KIM-1

En outre, la carte comporte un clavier, 6 LED, une interface magnétophone et une interface télétype. Elle peut être étendue à l'extérieur, grâce à deux connecteurs nez-de-carte, appelés respectivement connecteur d'extension et connecteur d'application, comme le montre la figure 3-3. La carte d'implantation-mémoire du système est représentée sur la figure 3-4. Les figures 3-5 et 3-6 donnent la liste des signaux des deux connecteurs du KIM.

L'organisation de cette carte est conforme à la description de notre système à 6502 standard (cf. fig. 3-1). Les détails des broches des connecteurs pourront être utiles aux lecteurs désireux de connecter les applications présentées dans ce livre à cette carte particulière.

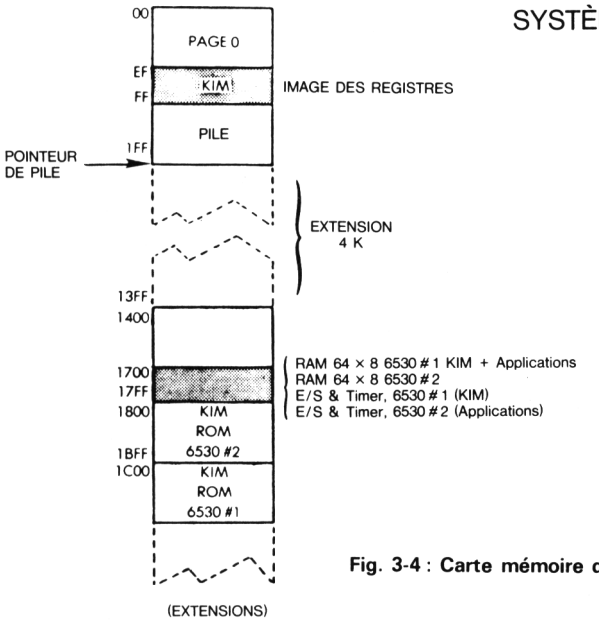


Fig. 3-4 : Carte mémoire du KIM-1

22	KB Col D	Z	KB Row 1
21	KB Col A	Y	KB Col C
20	KB Col E	X	KB Row 2
19	KB Col B	W	KB Col G
18	KB Col F	V	KB Row 3
17	KB Row 0	U	TTY PTR
16	PB5	T	TTY KYBD
15	PB7	S	TTY PTR RTRN (+)
14	PA0	R	TTY KYBD RTRN (+)
13	PB4	P	AUDIO OUT HI
12	PB3	N	+12V
11	PB2	M	AUDIO OUT LO
10	PB1	L	AUDIO IN
9	PB0	K	DECODE ENAB
8	PA7	J	K7
7	PA6	H	K5
6	PA5	F	K4
5	PA4	E	K3
4	PA1	D	K2
3	PA2	C	K1
2	PA3	B	K0
1	V_{SS} (GND)	A	V_{CC} (+5V)

Fig. 3-5 : Connecteur d'applications du KIM (A)

22	V_{SS} (GND)	Z	RAM/R/W
21	V_{CC} (+5)	Y	$\overline{\phi 2}$
20		X	PLL TEST
19		W	$\overline{R/W}$
18		V	R/W
17	SST OUT	U	$\phi 2$
16	K6	T	AB15
15	DB0	S	AB14
14	DB1	R	AB13
13	DB2	P	AB12
12	DB3	N	AB11
11	DB4	M	AB10
10	DB5	L	AB9
9	DB6	K	AB8
8	DB7	J	AB7
7	RST	H	AB6
6	NMI	F	AB5
5	O	E	AB4
4	IRQ	D	AB3
3	$\phi 1$	C	AB2
2	RDY	B	AB1
1	SYNC	A	AB0

Fig. 3-6 : Connecteur d'extension du KIM (E)

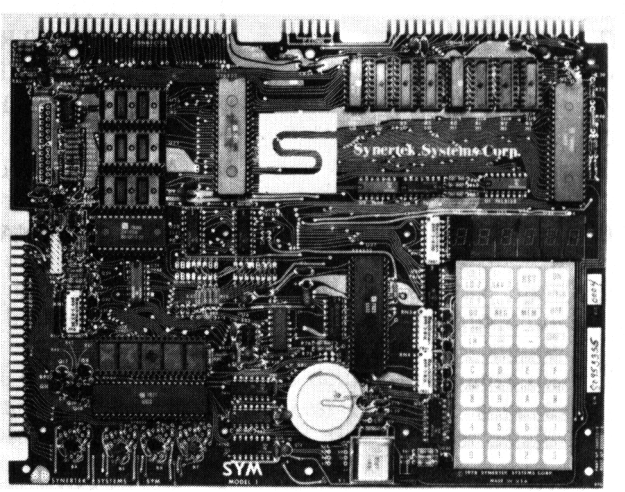


Fig. 3-7 : Le SYM

LE SYM-1

La carte SYM-1 a été introduite par Synertek Systems, comme version développée du KIM. (Cf. la photo de la figure 3-7 ; la figure 3-8 expose son organisation interne.)

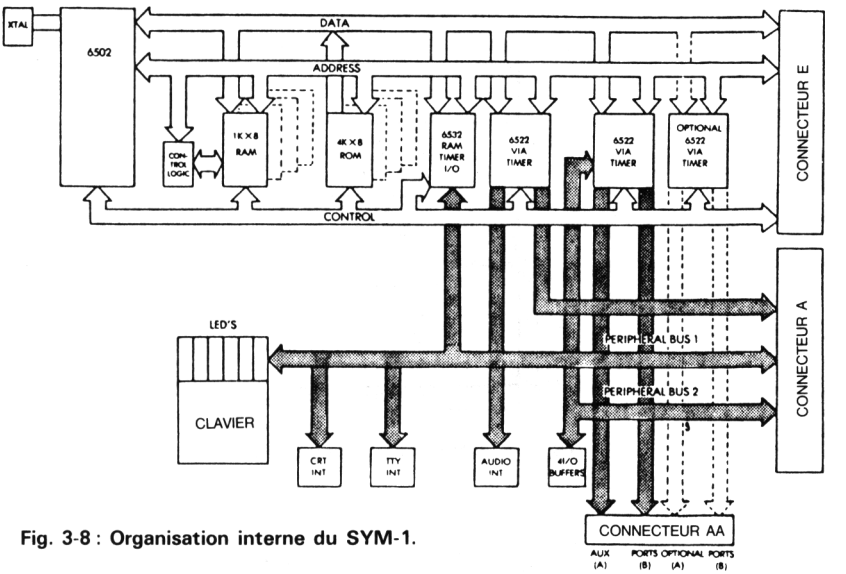


Fig. 3-8 : Organisation interne du SYM-1.

Comparé au KIM, le SYM1 présente une série de différences :

- Il est muni d'une ROM séparée de 4 K par 8. Une ROM de plus grande taille permet à un moniteur plus élaboré de résider sur la carte.
- Il est équipé de trois boîtiers d'entrée-sortie, plus élaborés que les deux dont dispose le KIM, et offre, par suite, plus de ressources et de ports d'E/S. Ces ports supplémentaires lui permettent de disposer d'un connecteur d'applications de plus que le KIM.
- Il dispose de possibilités d'entrées-sorties supplémentaires, telles que des entrées-sorties amplifiées et une interface oscilloscope.

Nous négligerons les autres différences, parce qu'elles concernent certains usages des deux cartes que nous n'aborderons pas dans cet ouvrage.

La carte d'implantation-mémoire apparaît figure 3-9, et la figure 3-10 montre une carte plus détaillée de la RAM. Les détails des trois connecteurs sont donnés respectivement par les figures 3-11, 3-12 et 3-13.

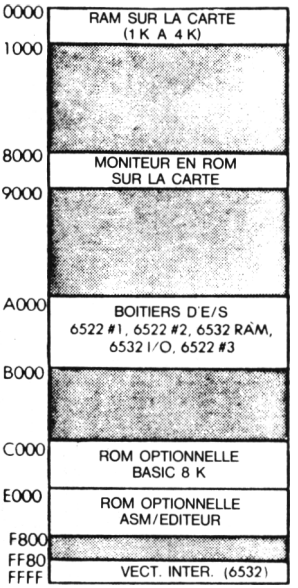


Fig. 3-3 : Carte mémoire du SYM

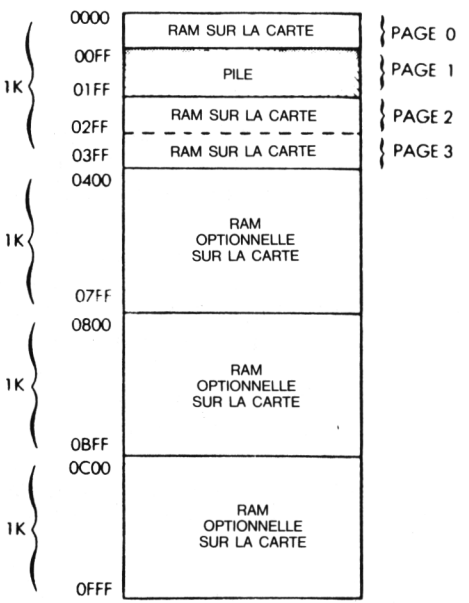


Fig. 3-10 : Carte d'implantation de la RAM du SYM

1	SYNC	A	AB0
2	RDY	B	AB1
3	ϕ1	C	AB2
4	$\overline{IRQ}$	D	AB3
5	SO	E	AB4
6	$\overline{NMI}$	F	AB5
7	$\overline{RES}$	H	AB6
8	DB7	J	AB7
9	DB6	K	AB8
10	DB5	L	AB9
11	DB4	M	AB10
12	DB3	N	AB11
13	DB2	P	AB12
14	DB1	R	AB13
15	DB0	S	AB14
16	$\overline{I8}$	T	AB15
17	DBOUT (1)	U	ϕ2
18	$\overline{POR}$	V	R/W
19	inutilisé	W	$\overline{R/W}$
20	inutilisé	X	AUD TEST
21	+ 5V	Y	$\overline{\phi 2}$
22	GND	Z	RAM - R/W

Fig. 3-11 : Connecteur d'extension du SYM (E)

1	GND	A	+ 5V
2	APA3	B	$\overline{00}$
3	APA2	C	$\overline{04}$
4	APA1	D	$\overline{08}$
5	APA4	E	$\overline{0C}$
6	APA5	F	$\overline{10}$
7	APA6	H	$\overline{14}$
8	APA7	J	$\overline{IC}$
9	APB0	K	$\overline{18}$
10	APB1	L	Audio In

Fig. 3-12 : Connecteur d'applications du SYM (A)

11	APB2	M	Audio Out (LO)
12	APB3	N	RCN-1 (1)
13	APB4	P	Audio Out (HI)
14	APA0	R	TTY KB RTN (+)
15	APB7	S	TTY PTR (+)
16	APB5	T	TTY KB RTN (−)
17	KB ROW 0	U	TTY PTR (−)
18	KB COL F	V	KB ROW 3
19	KB COL B	W	KB COL G
10	KB COL E	X	KB ROW 2
21	KB COL A	Y	KB COL C
22	KB COL D	Z	KB ROW 1

(1) : changeable par cavalier

Fig. 3-12 : Connecteur d'applications (suite)

1	GND	A	+5V
2	$-V_N$	B	$+V_P$
3	2 PA 1	C	2 PA 2
4	2 CA 2	D	2 PA 0
5	2 CB 2	E	2 CA 1
6	2 PB 7	F	2 CB 2
7	2 PB 5	H	2 PB 6
8	2 PB 3	J	2 PB 4
9	2 PB 1	K	2 PB 2
10	2 PA 7	L	2 PB 0
11	2 PA 5	M	2 PA 6
12	2 PA 3	N	2 PA 4
13	RES	P	3 CA 1
14	3 CB 1	R	SCOPE
15	3 PB 2	S	3 PB 3
16	3 PB 0	T	3 PB 1
17	3 PA 6	U	3 PA 7
18	3 PA 3	V	3 PA 0
19	3 PA 4	W	3 PA 1
20	3 PA 5	X	3 PA 2
21	3 PB 5 (B)	Y	3 PB 4 (B)
22	3 PB 7 (B)	Z	3 PB 6 (B)

(B) : Amplifié

Fig. 3-13 : Connecteur auxiliaire d'applications du SYM (AA)

Ab00	ORB (PBO A P87)
Ab01	ORA (PAO A PA7)
Ab02	DDR B
Ab03	DDR A
Ab04	T1L-L/T1C-L
Ab05	T1C-H
Ab06	T1L-L
Ab07	T1L-H
Ab08	T2L-L/T2C-L
Ab09	T2C-H
Ab0A	SR
Ab0B	ACR
Ab0C	PCR (CA1,CA2,CB2,CB1)
Ab0D	IFR
Ab0E	IER
Ab0F	ORA

b = 0 VIA # 1.
b = 8 : VIA # 2.
b = C : VIA # 3.

Fig. 3-14 : Adressage des 6522 du SYM

A41F	TIMER – 1024
A41E	TIMER ÷ 64T
A41D	TIMER ÷ 8T
A41C	TIMER ÷ 1T
	NON DISP
A407	(E) DET FRONT (L) ETAT
A406	(E) DET FRONT (L) TIMER
A405	(E) DET FRONT (L) ETAT
A404	(E) DET FRONT (L) TIMER
A403	DDRB
A402	ORB
A401	DDRA
A400	ORA

Fig. 3-15 : Adressage du 6532 du SYM

Les adresses utilisées pour les 6522 sont données par la figure 3-14, et celles du 6532 par la figure 3-15.

Nous présentons ci-dessous deux détails d'implantation, qui seront, avec d'autres, utilisés ultérieurement dans certains programmes d'application.

La figure 3-16 montre les quatre sorties amplifiées disponibles sur PB4-PB7 du 6522 n° 3, et la figure 3-17 la connection des LED et du clavier.

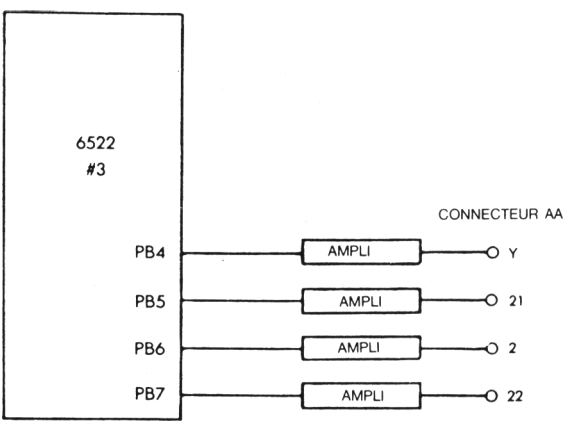


Fig. 3-16 : Les quatre sorties amplifiées du SYM

L'AIM 65

L'AIM 65 (1), développé par Rockwell International, est constitué de deux cartes. La première est le micro-ordinateur, équipé d'une imprimante 20 colonnes à matrice de points, et d'un affichage alphanumérique 20 caractères. La seconde supporte un clavier alphanumérique complet, relié directement à la première. L'imprimante fonctionne à plus de 120 lignes minute, et utilise une matrice de points 5 × 7, pour imprimer le jeu complet de 64 caractères ASCII (majuscules seulement). Dans sa version minimum, l'AIM 65 est muni d'un moniteur étendu (8 K), de 1 K de RAM, de deux 6522, d'un 6532 auxquels s'ajoutent les interfaces habituelles (télétype, deux magnétos à cassettes et, bien sûr, l'interface clavier). Des boîtiers supplémentaires peuvent facilement

(1) Voir fig. 3-18.

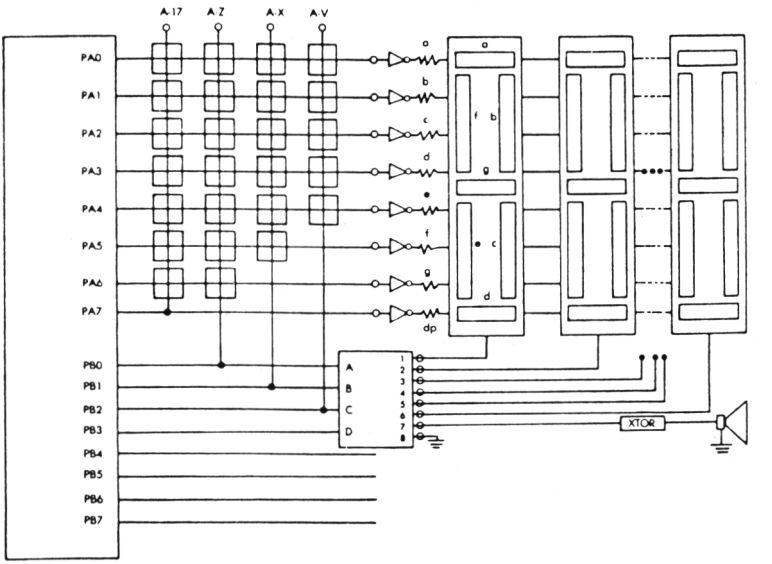


Fig. 3-17 : Connexions du clavier et des LED du SYM

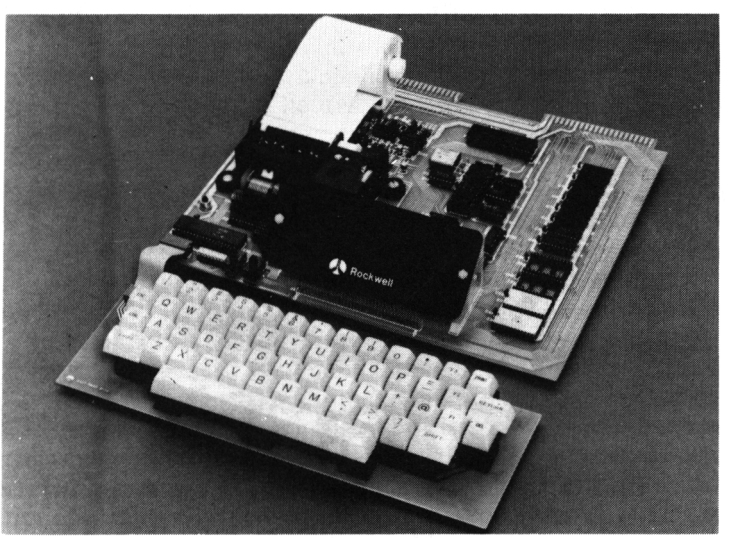


Fig. 3-18 : L'AIM 65 est une carte a micro-imprimante et clavier complet.

être ajoutés sur la carte. En outre, le connecteur d'application utilisateur est identique à ceux des cartes précédentes. Un utilisateur, développant des applications pour cette carte particulière, n'aura donc à modifier les programmes présentés ici, que pour s'adapter aux adresses attribuées aux PIO de l'AIM 65.

AUTRES CARTES

D'autres cartes sont fabriquées par divers constructeurs. Citons, parmi eux, Ohio Scientific.

En règle générale, toutes les cartes à 6502 obéissent à la description de notre « système standard ». Dans la mesure où ces cartes utilisent les mêmes boîtiers d'entrées-sorties (1), les programmes présentés dans ce livre n'ont pratiquement pas besoin d'être modifiés. On corrigera simplement les adresses des PIO, et l'absence possible de certaines lignes d'E/S spécifiques.

La figure 3-19 montre la compatibilité entre les connecteurs A et E du KIM, du SYM et de l'AIM. On observera, connectées à un KIM par le connecteur A, deux expériences décrites au chapitre VI : un clavier hexadécimal et une micro-imprimante. La carte verticale, à gauche de l'alimentation, est une extension-mémoire dynamique de 16K, connectée par le connecteur E. Les mêmes connecteurs pourraient, sans modification, être branchés respectivement sur les connecteurs A et E d'un SYM ou d'un AIM.

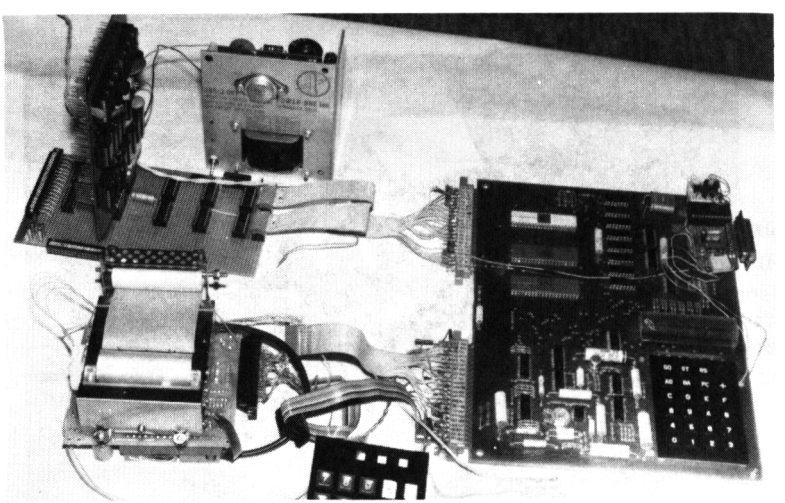


Fig. 3-19 : Compatibilité des connecteurs du KIM/SYM/AIM.

(1) Ce qui est presque toujours le cas, tant ces boîtiers présentent d'avantages.

4

TECHNIQUES ÉLÉMENTAIRES

INTRODUCTION

Connecter une carte à 6502 à des dispositifs d'entrée-sortie élémentaires : tel sera, à présent, notre objectif.

En sortie, nous connecterons une série de dispositifs simples, tels que des diodes émettrices de lumière (LED), des relais ou un haut-parleur, et en entrée, un ensemble d'interrupteurs.

Ces ressources nous permettront, ensuite, de commencer à développer des programmes d'applications simples : un générateur de Morse, une horloge 24 heures, un programme de surveillance d'une habitation, et même un composeur de numéros de téléphone. D'autres applications, découlant directement de ces techniques, sont également possibles. Nous en présenterons quelques-unes : une sirène, un testeur d'impulsions, un programme musical, un jeu mathématique.

Dans le chapitre suivant, nous développerons des programmes plus complexes, utilisant conjointement ces dispositifs d'entrée-sortie élémentaires, et d'autres, plus élaborés.

Peu de composants sont nécessaires pour réaliser la carte d'application de ce chapitre, comme le montre la photographie de la carte réelle (fig. 4-0). Très bon marché, ils sont en vente dans tous les magasins d'électronique. Nous conseillons vivement au lecteur de se les procurer, et de les monter conformément à notre description, afin de mettre en œuvre, concrètement, les programmes que nous lui soumettons. Mais cela suppose, naturellement, qu'il puisse avoir accès à une carte à 6502.

Nous utiliserons, dans la première partie, la configuration hardware de la carte SYM, et celle du KIM dans la deuxième partie. Néanmoins, tous les programmes devraient pouvoir être réalisés avec n'importe quelle autre carte 6502, au prix de simples aménagements de détail (cf. chap. III).

Les programmes proposés sont simples, mais ils supposent une compréhension élémentaire des instructions du 6502 (1).

Voici la liste des composants nécessaires aux programmes d'application :

Carte pastillée et percée	(1)
Interrupteurs	(4)
Ampli de LED	(1)
LED	(1 ou plus)
Relais 12 V	(3)
Haut-parleur	(1) d'impédance élevée de préférence
Résistance variable	(1)
Résistances	
Prise secteur mâle	(1)
Prises secteur femelles	(2)

La connection hardware des différents composants de la carte sera expliquée avec chaque application.

Il n'est pas rigoureusement indispensable d'assembler la carte d'application pour lire ce chapitre, et les suivants. Mais il serait regrettable de traiter seulement sur le papier les nombreux exercices qui y sont suggérés. Une véritable habileté en programmation s'acquiert, en premier lieu, par l'expérience réelle. Le lecteur est donc, encore une fois, encouragé à programmer le plus rapidement possible sur du matériel réel. Soit avant, soit après, la lecture de ce livre.

Notre but est d'enseigner les techniques d'interface hardware et software élémentaires requises pour connecter n'importe quelle carte 6502 à des dispositifs externes simples. A la fin de ce chapitre, vous devriez savoir utiliser les principales ressources des boîtiers d'entrées-sorties, et transcrire des programmes de surveillance et de commande des dispositifs d'entrée-sortie. Nous nous appuierons sur cette connaissance, dans le prochain chapitre, pour développer des applications industrielles et domestiques plus complexes.

---

(1) Cf. le précédent ouvrage de cette série, référence 331 : *Programmation du 6502.*

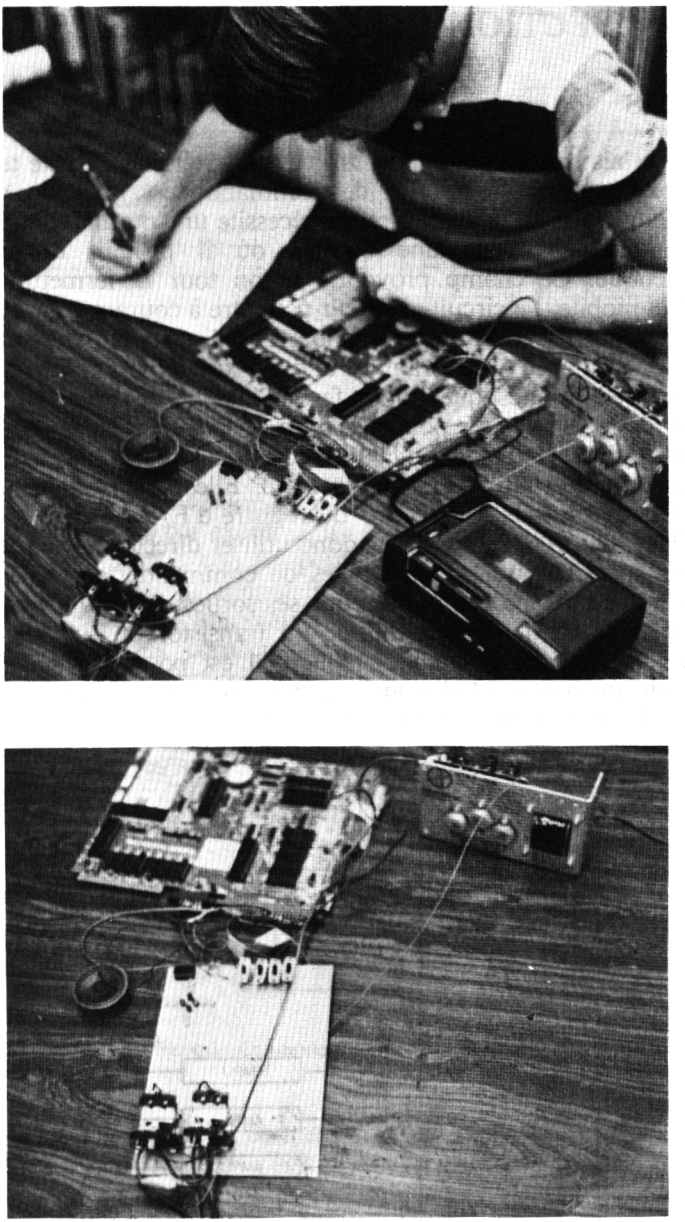


Fig. 4-0 : Système complet avec alimentation, carte microordinateur, magnétophone et carte d'applications.

SECTION 1 : LES TECHNIQUES

RELAIS

Un relais sert à commander un circuit extérieur à haute tension ou fort courant : le circuit de commande est isolé du circuit extérieur grâce au relais. Le relais nécessite un courant continu. Le courant circule dans une bobine, où il produit un champ magnétique. Ce champ provoque à son tour la fermeture d'un contact mobile. Le circuit extérieur peut être à courant continu (CC) ou alternatif (CA). Pour commander des dispositifs extérieurs qui fonctionnent avec un courant ou une tension considérables, comme les appareils domestiques, nous ferons appel à des relais.

La carte SYM est pourvue d'un dispositif spécial pour les appareils haute tension ou à fort courant. On dispose de quatre lignes de sortie amplifiées sur la carte. Elles sont respectivement connectées aux bits 4, 5, 6 et 7 du registre d'E/S B du PIO (6522-U29) (cf. fig. 4-1). Nous allons donc utiliser directement ces quatre sorties spéciales, qui sont capables de commander un relais. Sur toute autre carte ne comportant que ses sorties de PIO (comme, par exemple, le KIM), il faut utiliser un transistor ou un ampli. La figure 4-2 montre de quelle manière il est possible d'utiliser un sextuple inverseur 7404 pour commander trois relais externes, à partir de deux lignes de sortie d'un 6530.

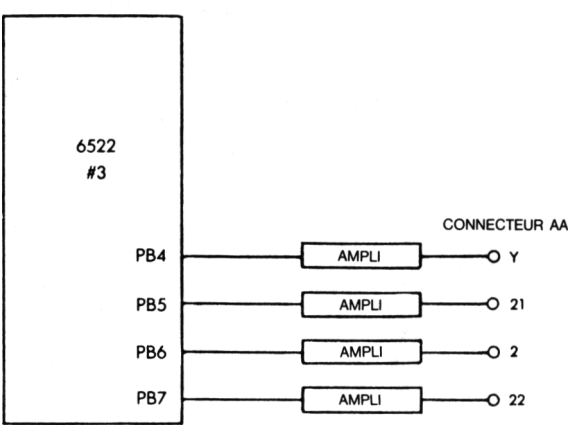


Fig. 4-1 : Amplis d'entrées-sorties

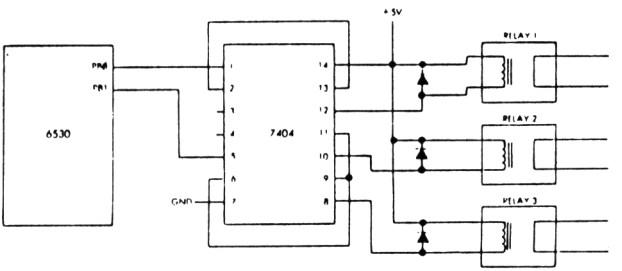


Fig. 4-2 : Interface 6530 - Relais

Interface hardware

Le schéma de connexion d'un relais unique apparaît figure 4-3. Ce relais peut être, par exemple, un relais 12 V, avec une bobine de 50 à 500 ohms. Le contact peut être SPST (Single Pole Single Throw = interrupteur unipolaire) ou SPDT (Single Pole Double Throw = inverseur unipolaire) à 10-15 ampères. Le courant admissible par les contacts du relais doit être suffisant pour s'accommoder de l'appareil extérieur qu'on lui connecte. La plupart des appareils ménagers ne demandent pas plus que 10-15 A. Les spécifications que nous venons d'évoquer devraient donc être suffisantes pour les applications domestiques.

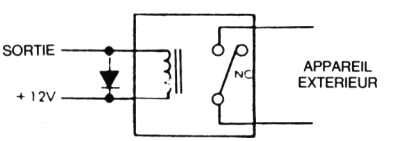


Fig. 4-3 : Connexion d'un relais simple

Notez, sur la figure, la connexion d'une diode de protection en parallèle avec la bobine. Il s'agit d'une précaution importante, quel que soit le relais, pour éviter d'endommager le tampon du PIO ou l'amplificateur. Un pic de tension inverse se produit à chaque fois qu'on ferme le relais. Toute diode qui supporte la tension peut être utilisée. Par exemple, une 1N914 devrait être suffisante pour nos besoins.

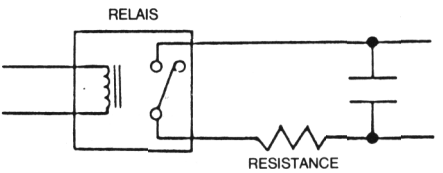


Fig. 4-4 : Protections du côté de l'appareil extérieur

Du côté de l'appareil extérieur, deux précautions sont conseillées. Il peut être judicieux de placer un condensateur en parallèle avec la sortie, pour absorber la discontinuité due à la fermeture du contact. On augmente ainsi la longévité des contacts du relais. En outre, dans l'hypothèse d'une forte consommation de courant, il est nécessaire de placer une résistance en série (cf. fig. 4-4).

On peut connecter exactement de la même manière un relais bipolaire (= 2 contacts). Le schéma de connexion apparaît figure 4-5. Un tel relais est capable de commuter simultanément deux circuits séparés indépendants.

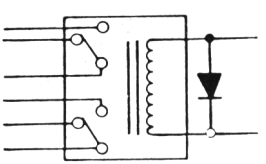


Fig. 4-5 : Connexion d'un relais bipolaire, double contact (= inverseur)

Considérons maintenant une application pratique. Nous allons connecter deux relais R1 et R2, respectivement aux bits 6 et 7 du port B du PIO du SYM. Ces deux relais commanderont des appareils à courant alternatif. Dans le cas le plus simple, nous supposerons qu'il s'agit de deux lampes indépendantes, ce qui nous permettra de tester aisément le programme, en vérifiant que les lampes sont correctement allumées ou éteintes. On pourrait, naturellement, remplacer une lampe par n'importe quel appareil ménager qui ne surchargerait pas le relais. Le schéma de branchement apparaît sur la figure 4-6.

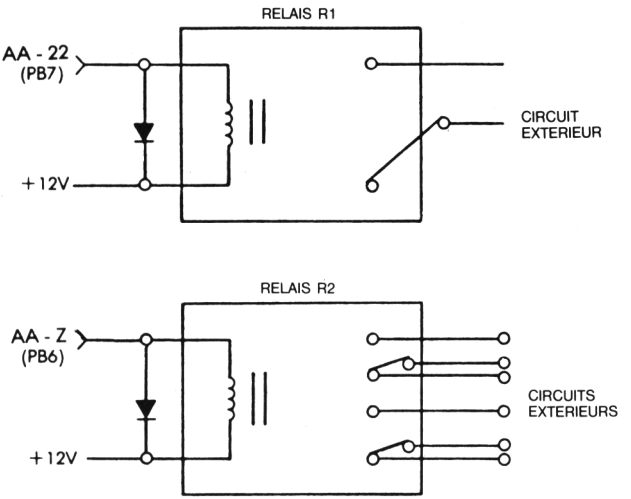


Fig. 4-6 : Connexion de deux relais au PIO

Examinons les figures 3-11, 3-12 et 3-13, qui font apparaître les broches des trois connecteurs du SYM. Nous voyons que les quatre sorties amplifiées, appelées PB4, PB5, PB6 et PB7, sont disponibles, respectivement, sur les broches Y, 21, Z et 22. Les contacts marqués PB5 et PB7 sur notre illustration doivent donc simplement être reliés à la broche convenable du « connecteur auxiliaire d'applications ».

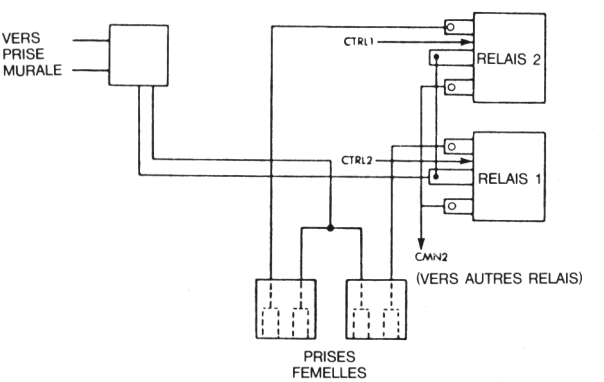


Fig. 4-7 : Circuit externe des relais

Du côté du circuit externe des relais, on utilisera une prise secteur, qui sera branchée dans une prise murale et alimentera les deux prises femelles contrôlées par le micro-ordinateur. Ces deux prises seront connectées aux relais, comme le montre la figure 4-7. Elles seront alimentées en parallèle par la prise secteur. Néanmoins, l'une ou l'autre pourra être alimentée indépendamment, sous contrôle du micro-ordinateur. Réalisons maintenant le programme de commande de ces relais.

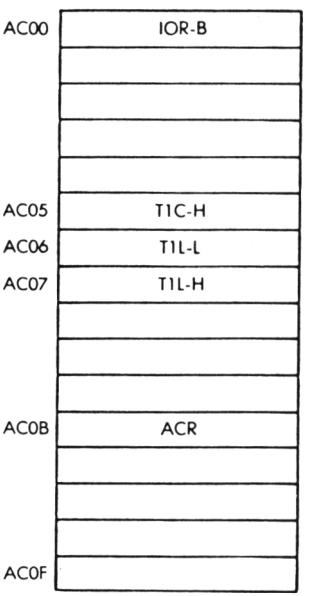


Fig. 4-8 : Carte-mémoire du 3e 6522 du SYM

Interface software

Chacun des deux circuits connectés aux relais R1 et R2 sera alimenté lorsque le relais correspondant sera, de son côté, activé. Ce qui adviendra en plaçant à 1 le bit de commande correspondant. En examinant la figure 4-8, on voit que le port B du 3e 6522 est situé à l'adresse-mémoire AC00. Le contenu de l'adresse AC00 est illustré sur la figure 4-9. Mettons maintenant les relais en marche et à l'arrêt.

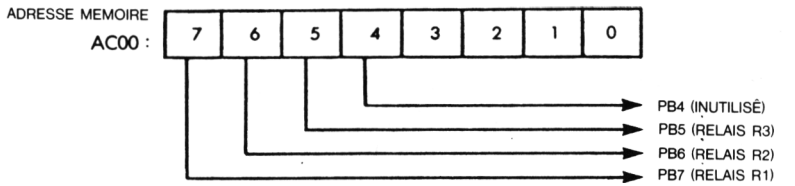


Fig. 4-9 : Port B du 6522 n° 3

Le port B sera d'abord configuré en sortie. Pour simplifier, nous allons spécifier que tous les bits de 0 à 7 sont des sorties, bien qu'ici nous n'utilisions que les bits 5, 6 et 7. La convention pourrait être différente dans une autre application. On se rappellera, depuis le chapitre 2, que, pour spécifier le mode d'utilisation des lignes d'entrée-sortie, il faut mettre un zéro ou un 1 dans le bit correspondant du registre direction. Un 1 spécifie une sortie, un zéro, une entrée. Charger exclusivement des 1 dans le registre direction implique que tous les bits seront utilisés en sortie.

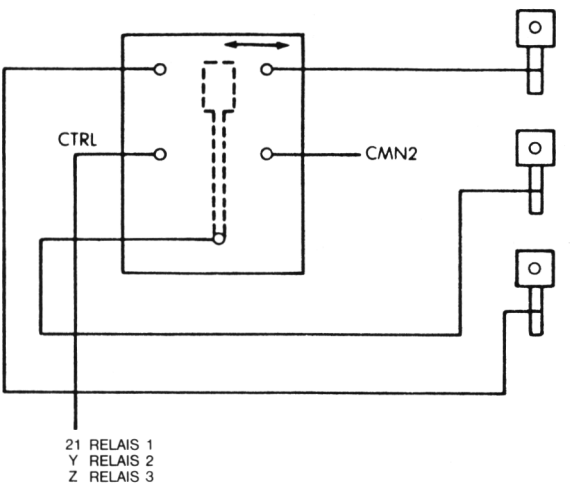


Fig. 4-10 : Détail de la connexion d'un relais sur la carte d'applications.

Remarque : En programmation, il est toujours bienvenu de rendre les choses aussi simples et cohérentes que possible. Comme nous supposons, pour le moment, qu'aucun autre dispositif n'est connecté aux autres lignes du port B, il est plus sûr de configurer toutes les lignes en entrées ou en sorties.

La mise en sortie de tous les bits s'effectue grâce aux deux instructions :

```
LDA   #$FF    CHARGEMENT IMMÉDIAT DE A
                AVEC 11111111
STA   $AC02   RANGE A A L'ADRESSE AC02 HEXA
```

On peut vérifier sur la figure 4-8 que AC02 est l'adresse du registre direction du port B du 6522 n° 3. "FF" en hexadécimal est l'équivalent de "11111111" en binaire. Alimentons maintenant le relais connecté à PB6 :

```
LDA   $AC00   LIT LA VALEUR ACTUELLE DE PB
ORA   #$40    FORCE LE BIT 6 A 1
STA   $AC00   SORTIE
```

La première instruction permet de lire l'état actuel du port B. Comme plusieurs dispositifs ou relais peuvent être simultanément connectés au port B, on ne peut se contenter d'écrire un motif comme "01000000" sur le port B ; cela activerait le relais connecté à PB6, mais arrêterait tous les autres !

Il faut en conséquence, lire l'état actuel de PB et changer le seul bit PB6. Ce changement est réalisé par l'instruction de OU logique (ORA), 2e instruction de notre programme. Le OU logique laisse inchangés tous les bits, et il force un 1 à l'emplacement spécifié. Si nous voulions activer PB7 au lieu de PB6, nous utiliserions le motif "80" hexadécimal, au lieu de "40". Enfin, le motif binaire résultant est écrit à l'adresse AC00, qui correspond à PB ; le relais est alors activé.

Exercice 4-1 : *Ecrire le programme de trois instructions qui active simultanément les relais connectés à PB6 et PB7.*

Mettons maintenant à l'arrêt le relais connecté à PB6 :

```
LDA   $AC00   LIRE L'ÉTAT ACTUEL DE PB
AND   #$BF    METTRE LE BIT 6 A 0
STA   $AC00   ÉCRIRE LA VALEUR RÉSULTANTE EN PB
```

L'instruction de ET logique sert à forcer un "0" à la position voulue. Tous les autres bits sont inchangés. ("BF" en hexadécimal correspond à "10111111" en binaire.)

Note : On utilise traditionnellement l'instruction AND pour mettre à zéro un bit déterminé. Cependant, le même résultat peut être obtenu à l'aide de l'instruction EOR. Le programme reste le même, à l'exception de l'instruction AND qui devient :

```
EOR    #$40
```

L'avantage est que le motif binaire utilisé pour désactiver ou pour activer est le même. Cela élimine un risque d'erreur.

Le lecteur devra naturellement vérifier la légitimité de cette manière de forcer un 0. Pourquoi est-ce légitime ? Parce que le OU exclusif de 1 avec 1 est 0.

Si le bit 6 est à 1, le motif "40" le force à 0. Les autres bits sont inchangés.

Vérification

Il convient maintenant de s'assurer que ces instructions simples sont, en effet, suffisantes pour activer et désactiver nos relais, en connectant à ces derniers deux lampes ou deux appareils, et en tapant les instructions au clavier. On vérifiera ensuite si les lampes sont allumées ou éteintes. Comme le clavier exige que les entrées soient sous forme hexadécimale, voici l'équivalent hexadécimal des deux programmes ci-dessus :

Pour activer le relais :

```
AD 00 AC
09 MOTIF (MOTIF représente un motif de 8 bits)
8D 00 AC
```

Pour désactiver le relais :

```
AD 00 AC
49 MOTIF
8D 00 AC
```

Si vous disposez d'un micro-ordinateur, nous vous conseillons maintenant de taper ces deux programmes et de vérifier qu'ils fonctionnent correctement.

INTERRUPTEURS

Il existe deux principaux types d'interrupteurs : l'interrupteur (SPST) et l'inverseur (SPDT). Le branchement d'un interrupteur apparaît figure 4-11. Avec le branchement indiqué, l'interrupteur est dans l'état logique "1" lorsque le contact est ouvert, et dans l'état "0" lorsque le contact est fermé. Dans le cas contraire, il suffirait d'inverser les branchements.

Le branchement d'un inverseur (interrupteur à deux positions) est illustré par la figure 4-12. Le branchement est évident : une des positions sera l'état logique "1", et l'autre le "0".

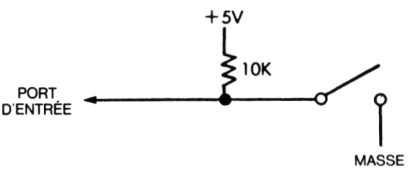


Fig. 4-11 : Branchement d'un interrupteur

Fig. 4-12 : Branchement d'un inverseur

Branchement de quatre interrupteurs

Nous allons utiliser les lignes 1, 2, 3 et 4 du port B du 6522 en entrées, pour déterminer l'état des interrupteurs extérieurs. Le branchement réel apparaît figure 4-13. Examinons le programme.

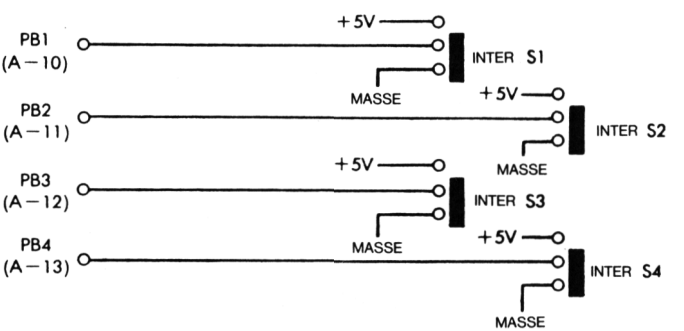


Fig. 4-13 : Branchement de 4 inverseurs sur le SYM

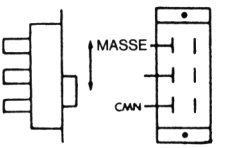


Fig. 4-14 : Inverseur

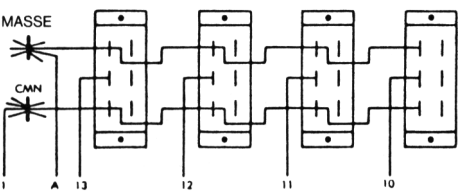


Fig. 4-15 : Détail du branchement des 4 inverseurs

Interface software

Il est tout d'abord nécessaire de configurer PB1, PB2, PB3 et PB4 en entrée. Ce résultat est obtenu en chargeant le motif binaire approprié à l'adresse « A002 », registre direction du port B.

```
LDA   #$E0     MET LES BITS 01234 EN ENTRÉE
STA   $A002
```

On utilisera le motif '5E0" pour mettre les lignes 0, 1, 2, 3 et 4 en entrée et les lignes 5, 6 et 7 en sortie (elles peuvent être connectées à des relais). "E0" est l'équivalent hexadécimal de "11100000". Chaque 0 définit une entrée ; chaque 1 définit une sortie. On pourrait aussi utiliser "E1".

Lisons maintenant l'état de l'interrupteur, et branchons-nous à l'adresse convenable en fonction de cet état.

```
LDA   #INTPTR  "02" pour S1, "04" pour S2, "08" pour S3,
               "10" pour S4
BIT   $A000    A000 EST L'ADRESSE DE PB
BEQ   ADRESSE  SE BRANCHE A L'ADRESSE SPÉCIFIÉE
               SI L'INTERRUPTEUR EST A 0
```

Inversement, si l'interrupteur est à 1, on remplacera la dernière instruction BEQ par un BNE, pour se brancher à l'adresse spécifiée.

Vérification du programme sur le micro-ordinateur

Le code hexadécimal du programme ci-dessus est :

A9 INTPTR
2C 00 A0
F0 ADR ou D0 ADR

HAUT-PARLEUR

On peut connecter un haut-parleur extérieur directement à une broche d'un des PIO. Sur les 6522, l'état de PB7 peut être commandé par l'un des temporisateurs. Le temporisateur sera utilisé pour générer un son de fréquence déterminée. La position la plus favorable pour connecter le haut-parleur sera donc PB7. Le schéma de branchement apparaît sur la figure 4-16.

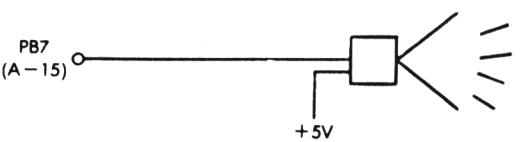


Fig. 4-16 : Branchement du haut-parleur

Lorsqu'on utilise une sortie amplifiée du SYM (6522 n° 3), il faut placer une résistance en série avec le haut-parleur pour limiter le courant de sortie. Au lieu de connecter directement le haut-parleur à une broche de sortie du PIO, on peut utiliser le circuit de la figure 4-17 pour obtenir un son plus fort.

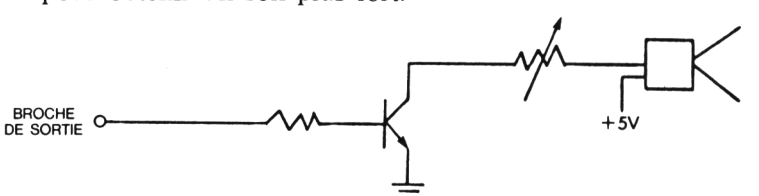


Fig. 4-17 : Obtention d'un son plus fort

Précaution : Une résistance variable a été placée sur la figure 4-17 pour raison de commodité. Il vaut mieux, néanmoins, se garder de la mettre à 0, car elle aurait alors toute chance d'être détruite, ce qui entraînerait la destruction consécutive du transistor de sortie correspondant. Ceci s'applique également au SYM.

Interface software

On peut obtenir un son du haut-parleur en envoyant simplement des 0 et des 1 à la fréquence désirée. Le son sera moins bon que s'il émanait d'un instrument de musique parce que produit par un signal carré. Il sera toutefois suffisant pour nos besoins, et pourra être clairement identifié par sa fréquence. Nous allons maintenant construire une application pratique.

CODEUR MORSE

Nous développons ici un programme capable de générer le code Morse correspondant à chaque lettre de l'alphabet. Ce programme agira, pour vérification, sur un haut parleur. Il commandera, en outre, un dispositif extérieur permettant, par exemple, de transmettre le code Morse sur une ligne de télécommunications.

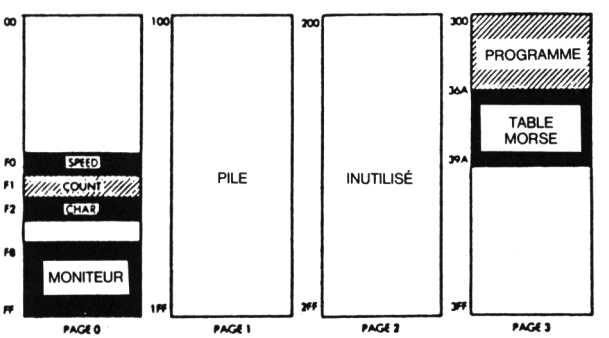


Fig. 4-18 : Allocation mémoire pour le programme de Morse

Les conventions établies pour ce programme sont les suivantes :

Le programme lui-même résidera dans la page 3 de la RAM, commençant à l'adresse 300. L'illustration en est donnée par la figure 4-18. Il contiendra une table d'équivalence des codes Morse, qui servira à obtenir le motif binaire approprié, correspondant à un caractère ASCII donné. Nous montrerons plus loin comment constituer cette table.

D'autre part, la vitesse de transmission sera ajustable par l'intermédiaire de la variable SPEED, rangée en page 0 à l'adresse

F0 (cf. fig. 4-18). Chaque unité de temps (par exemple, la durée d'un "point" en code Morse), sera exprimée en milli-secondes. La valeur 100 dans la variable SPEED permettra d'obtenir une durée de 1/10 de seconde.

On supposera que les variables CHAR et SPEED ont un contenu défini avant le démarrage du programme, et que l'accumulateur contient le premier caractère à transmettre. Un sous-programme pourra appeler ce sous-programme de façon répétée, afin de transmettre une chaîne de caractères. Chaque fois qu'il appellera le transmetteur Morse, le programme devra déposer un caractère dans l'accumulateur.

Examinons maintenant l'algorithme utilisé pour transmettre le code Morse.

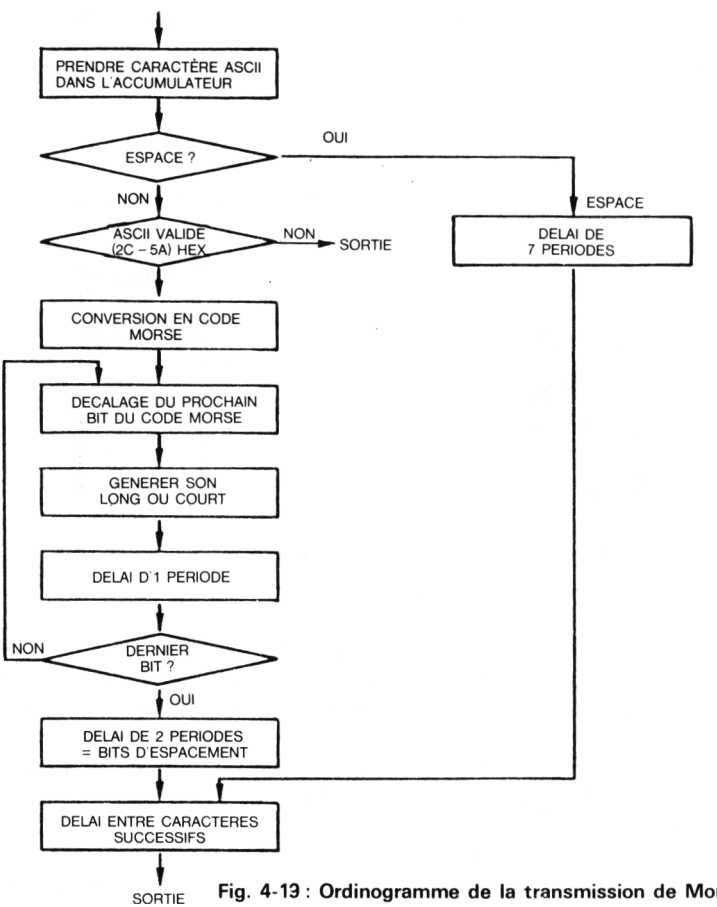


Fig. 4-13 : Ordinogramme de la transmission de Morse

Cet algorithme est illustré par l'ordinogramme de la figure 4-19. Le programme devra d'abord établir s'il s'agit d'un espace. Si oui, on ne générera aucun signal pendant 7 périodes, plus le délai entre caractères successifs. Sinon, on vérifiera que le caractère ASCII contenu dans l'accumulateur a un code hexadécimal valide.

Les codes valides doivent être compris entre "2C" et "5A" HEXA (Code ASCII à 7 bits). Sinon, on aura une sortie sur erreur. Après validation du code caractère, le code ASCII doit être converti en son équivalent Morse. La technique sera expliquée plus tard.

Le codage du Morse consistera en un bit de "DÉPART" ("1") suivi d'un 0 pour chaque "." (point) et d'un 1 pour chaque " – " (trait). Tous les bits inutilisés d'un octet (à gauche du bit de départ) seront mis à 0. Cette conversion sera effectuée par le programme à l'aide d'une technique d'examen de table décrite ci-après. Nous admettrons pour le moment que la valeur binaire du code Morse a été obtenue.

Il s'agit ensuite de générer la suite de sons. On décalera à gauche, le contenu de l'accumulateur, jusqu'au bit de départ. Après détection de ce dernier, tout "0" sera interprété comme un "." et tout "1" comme un " – ", jusqu'au huitième bit. Pour chaque "0" décalé au-dehors on génèrera un son court, et pour chaque "1", un son long. La génération du son sera décrite en détail plus tard.

Ensuite, on insérera un temps d'attente d'une période avant de passer au bit suivant du code Morse, jusqu'au dernier (le 8e).

L'étape suivante consistera à générer un délai de 2 périodes. Cela correspond à des bits d' "espace" qui sont normalement insérés à la fin de toute transmission d'un caractère. Enfin on générera les délais d'une période qui séparent les caractères successifs.

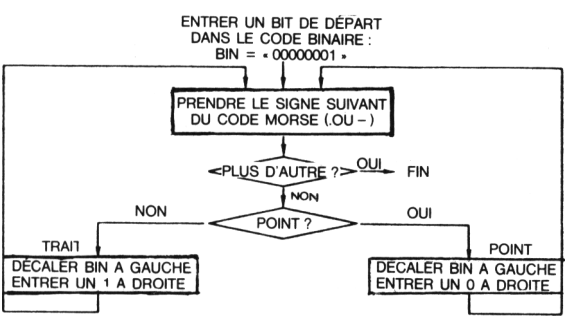


Fig. 4-20 : Conversion de Morse en binaire

La séquence est illustrée clairement sur l'ordinogramme de la figure 4-20 et le lecteur est encouragé à le vérifier. Revenons maintenant, en détail, sur les problèmes en suspens.

Conversion d'ASCII en Morse

Nous établirons ici une table de correspondance entre le caractère ASCII et la représentation binaire de son code Morse. Illustrons ceci par un exemple.

Le caractère "B" a pour code morse " – . . .". Chaque " – " sera codé comme "1", et chaque "." comme "0". L'équivalent binaire de " – . . ." sera donc "1000".

En outre, par convention, on ajoutera un bit de DÉPART (un "1") à gauche du code ainsi obtenu. Pour le moment le code qui en résulte pour B est donc : "11000". Pour finir chaque code morse sera contenu dans un mot de 8 bits. Les bits inemployés, à gauche du bit de départ, seront mis à 0. Le code 8 bits qui en résulte est donc : "00011000". En hexadécimal : "18".

La représentation hexadécimale du code Morse pour B est donc "18".

A titre d'exemple, la table ci-dessous montre l'équivalent hexadécimal de A, B, C et D. Une table d'équivalence complète pour tous les caractères Morse autorisés se trouve figure 4-22. L'ordinogramme correspondant à la technique décrite est donné figure 4-23.

Lettre	ASCII	Morse	binaire	hexadecimal
A	41	. —	00000101	05
B	42	— . . .	00011000	18
C	43	— . — .	00011010	1A
D	44	— . .	00001100	0C

Fig. 4-21 : Conversion d'ASCII en Morse

Nous avons maintenant établi une table d'équivalence pour tous les caractères ASCII. Nous l'appellerons « Table de Morse ». Elle sera rangée à la fin du programme (cf. fig. 4-18). Chaque fois que nous aurons besoin de l'équivalent Morse d'un caractère donné, nous accéderons à la ligne appropriée de la table pour y trouver le code binaire. Nous décrirons cette opération plus tard, en discutant le programme proprement dit.

Caractère	Code Morse	ASCII	valeur hexa
,	— — . . — —	2C	73
—	— . . . —	2D	31
.	. — . — . —	2E	55
/	— . . — .	2F	32
ø	— — — — —	3ø	3F
1	. — — — —	31	2F
2	. . — — —	32	27
3	. . . — —	33	23
4	. . . . —	34	21
5	. . . . .	35	2ø
6	—	36	3ø
7	— — . . .	37	38
8	— — — . .	38	3C
9	— — — — .	39	3E
:	libre (*)	3A	ø1
;	" "	3B	ø1
<	" "	3C	ø1
=	" "	3D	ø1
>	" "	3E	ø1
?	. . — — . .	3F	4C
@	libre	4ø	ø1
A	. —	41	ø5
B	— . . .	42	18
C	— . — .	43	1A
D	— . .	44	øC
E	.	45	ø2
F	. . — .	46	12
G	— — .	47	øE
H	. . . .	48	1ø
I	. .	49	ø4
J	. — — —	4A	17
K	— . —	4B	øD
L	. — . .	4C	14
M	— —	4D	ø7
N	— .	4E	ø6
O	— — —	4F	øF
P	. — — .	50	16
Q	— — . —	51	1D
R	. — .	52	øA
S	. . .	53	ø8
T	—	54	ø3
U	. . —	55	ø9
V	. . . —	56	11
W	. — —	57	øB
X	— . . —	58	19
Y	— . — —	59	1B
Z	— — . .	5A	1C

Fig. 4-22 : Table d'équivalence Morse

(*) à définir par l'utilisateur

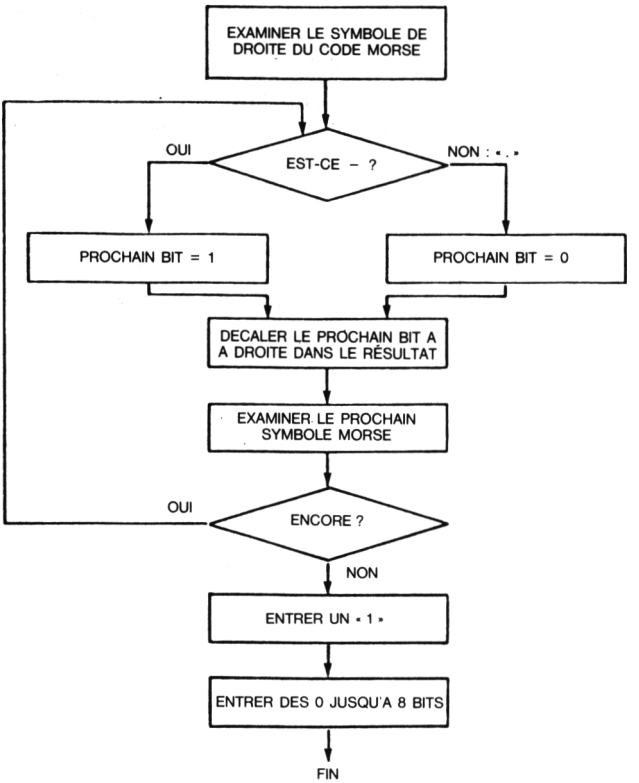


Fig. 4-23 : Ordinogramme de la génération du code Morse hexadécimal

Génération d'un son à l'aide du temporisateur

Notre prochain problème sera de générer un son de durée et fréquence données, à l'aide d'un temporisateur.

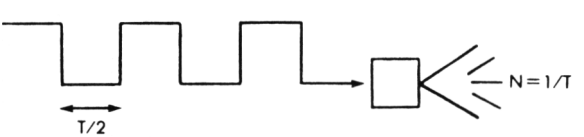
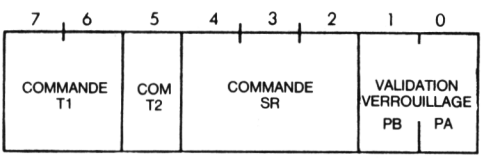


Fig. 4-24 : Un signal carré fait émettre un son par le haut-parleur

Le son sera produit par le haut-parleur, à qui on aura envoyé un signal carré de la fréquence voulue (cf. fig. 4-24). Le temporisateur peut générer ce signal carré automatiquement. Le résultat sera obtenu en positionnant les bits appropriés dans le registre ACR (cf. fig. 4-25), puis, tout simplement, en contrôlant la durée d'émission du son. Le diagramme des temps effectifs apparaît figure 4-26. $\varphi 2$, dans la partie supérieure de l'illustration, représente la phase 2 de l'horloge-système. Dans la plupart des systèmes à 6502 standards, l'horloge a une période de 1 μs. L'impulsion générée par le temporisateur apparaît sur la broche PB7. Elle dure N + 1,5 cycle, si N est la valeur initiale déposée dans le compteur. Pourquoi ? Parce que le compteur décrémente jusqu'à 0, et qu'il inverse le signal sur PB7, au départ d'un front descendant de l'horloge. Une illustration en est donnée par la figure 4-26. A ce même instant, une interruption est générée, mais elle ne sera pas utilisée ici.



Fg. 4-25 : Registre de commande auxiliaire du 6522

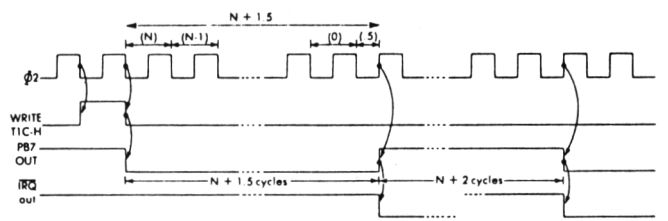


Fig. 4-26 : Diagramme des temps pour la production d'un son

Pour utiliser le temporisateur, il est nécessaire de déposer une valeur appropriée N dans son compteur. Mais toute écriture dans le compteur provoque immédiatement le début de la décrémentation. Il s'agit d'un registre 16 bits, qui ne peut être chargé par un seul transfert de données, depuis le microprocesseur : *une mémorisation est nécessaire.* Le temporisateur est donc muni d'un tampon interne 16 bits, appelé T1L, dont la partie basse sera dénommée T1L-L, et la partie haute T1L-H. La valeur N sera déposée dans T1L-L et T1L-H. A ce moment, le contenu des 16 bits est spécifié, mais rien n'a encore eu lieu. Pour démarrer le temporisateur, il est nécessaire de donner une commande spéciale, qui transférera le contenu du tampon dans le compteur proprement dit. C'est l'opération dite « écriture dans T1C-H », qui forme la 5e ligne de la figure 4-27 :

```
LDA    #VALBAS
STA    $A006          CHARGE TAMPON BAS
LDA    #VALHAUT
STA    $A007          CHARGE TAMPON HAUT
STA    $A005          TRANSFERT TAMPON → COMPTEUR = DÉPART
```

Fig. 4-27 : Programme d'utilisation du temporisateur T1

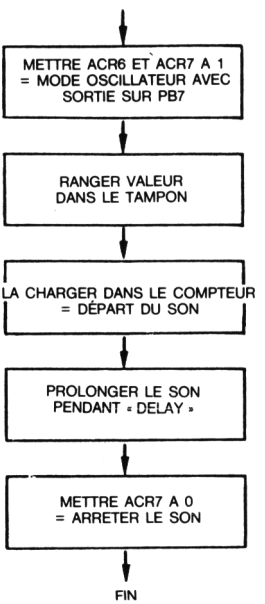


Fig. 4-28 : Production d'un son de durée déterminée avec le temporisateur 1

La séquence d'événements qu'il convient de déclencher pour pouvoir utiliser le temporisateur devrait maintenant être clairement perçue. Elle est décrite par l'ordinogramme de la figure 4-28. Il faut, en premier lieu, mettre les valeurs souhaitées dans les bits appropriés du registre ACR. Le temporisateur fonctionne en mode oscillateur, avec sortie d'un signal carré sur PB7. Ce résultat est obtenu en mettant à 1 les bits 6 et 7 de ACR (cf. fig. 4-29 et 4-30). On range, ensuite, dans le tampon la valeur voulue N. Cette valeur est transférée dans le compteur lui-même, ce qui a pour effet de démarrer la temporisation, et marque, par ailleurs, le début de l'émission du son recherché. Chaque fois que le compteur arrive à 0, il recharge automatiquement la valeur rangée dans le tampon. Le temporisateur va donc, à partir de ce moment, générer un signal carré d'une demi-période, égale approximativement à N + 2 cycles. Il s'agit là, en réalité, d'une valeur approchée, car la première alternance de l'impulsion dure N + 1,5 cycle, et les suivantes N + 2 cycles.

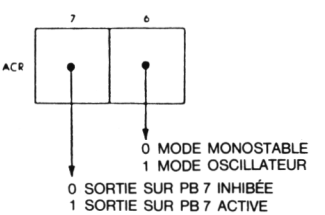


Fig. 4-29 : 6522 : contrôle de T1 par le registre de commande auxiliaire

ACR7 ACTIVATION SORTIE PB7	ACR6 MODE OSCILLATEUR	MODE
0	0 (MONOSTABLE)	Génère une interruption de time-out lorsque T1 est chargé. PB7 inactive.
0	1 (OSCILLATEUR)	Génère des interruptions continuelles. PB7 inactive.
1	0 (MONOSTABLE)	Génère une interruption et une impulsion sur PB7 à chaque chargement de T1 = monostable et impulsion de largeur programmable.
1	1 (OSCILLATEUR)	Génère des interruptions continuelles et un signal carré sur PB7.

Fig. 4-30 : Modes de fonctionnement du temporisateur 1 du 6522 sélectionnés par le registre de commande auxiliaire

```
LINE # LOC     CODE       LINE

0002  0000                ;THIS IS A SUBROUTINE WHICH ACCEPTS ASCII CHARACTERS
0003  0000                ;IN THE RANGE 2CH TO 5AH (PLUS 20H FOR SPACE) AND PLAYS
0004  0000                ;THEIR MORSE CODE EQUIVILENT ON A SPEAKER MODIFIED UP TO
0005  0000                ;R87, 6522-025.  IT ALSO TURNS ON AND OFF PB0, 6522-
0006  0000                ;025, AND WITH A SUITABLE DRIVER, THIS BIT CAN KEY A
0007  0000                ;TRANSMITTER.   A MAIN PROGRAM WILL CALL THIS SUBROUTINE
0008  0000                ;WITH THE ASCII CHARACTER IN THE ACCUMULATOR.
0009  0000                ;EXAMPLES OF THE MAIN PROGRAM WOULD BE ONE THAT
0010  0000                ;GETS INPUT FROM A TERMINAL AND SENDS MORSE CODE OUT
0011  0000                ;THROUGH THIS PROGRAM, OR A PROGRAM WHICH RANDOMIZES
0012  0000                ; A SERIES A CHARACTERS AND SENDS THEM FOR CODE PRACTICE.
0013  0000                ;THE FORMAT FOR THE MORSE CODE CAHRACTERS IN THE TABLE
0014  0000                ;IS : MOVING FROM LEFT TO RIGHT , THE FIRST HIGH
0015  0000                ;BIT (A ONE) IS THE START BIT, AND AFTER THIS ,
0016  0000                ;EACH ONE IS A DASH, AND EACH ZERO IS A DOT.
0017  0000                SPEED=$F0
0018  0000                COUNT=$F1
0019  0000                CHAR=$F2
0020  0000                      .=$300
0021  0300: C9 20         MORSE  CMP #$20        ;IF A SPACE, DO SPACE ROUTINE
0022  0302: F0 67                BEQ SPACE
0023  0304: C9 2C                CMP #$2C       ;SEE IF ASCII CODE
0024  0306: 90 4E                BCC EXIT       ;    IS LESS THEN 2CH, AND RETURN IF SO.
0025  0308: C9 5B                CMP #$5B       ;SEE IF ASCII CODE IS OVER
0026  030A: B0 4A                BCS EXIT       ;    5AH, AND RETURN IF SO
0027  030C: AA                   TAX            ;PUT CODE IN INDEX REGISTER
0028  030D: BD 45 03             LDA TABLE-$2C,X ;GET MORSE CHARACTER
0029  0310: A0 08                LDY #$8        ;NUMBER OF BITS TO BE ROTATED FROM ACCUMULATOR
0030  0312: 84 F1                STY COUNT
0031  0314: 0A            STARTB ASL A
0032  0315: C6 F1                DEC COUNT
0033  0317: 90 FB                BCC STARTB     ;SHIFT A UNTIL START BIT FOUND
0034  0319: 85 F2                STA CHAR
0035  031B: A5 F2           NEXT   LDA CHAR
0036  031D: 0A                   ASL A          ;NOW SHIFT OUT MORSE CODE (1=DASH, 0=DOT)
0037  031E: 85 F2                STA CHAR
0038  0320: A0 01                LDY #$1        ;DOT= 1 TIME PERIOD, DEFAULT TO DOT
0039  0322: 90 02                BCC SEND       ;IF CARRY CLEAR, DOT
0040  0324: A0 03                LDY #$3        ;ELSE DASH    (3 TIME PERIODS)
0041                      ; THIS SECTION SENDS A HIGH OUTPUT FOR (Y REGISTER ) NU
0042                      ;OF TIME PERIODS, AND THEN A LOW FOR 1 TIME PERIOD.
0043  0326: A9 C0         SEND   LDA #$C0
0044  0328: 8D 0B A0             STA $A00B      ;SET TIMER  MODE TOFREE RUNNING MODE
0045  032B: A9 00                LDA #$0        ; THIS VALUE,
0046  032D: 8D 06 A0             STA $A006
0047  0330: A9 04                LDA #$04       ; AND THIS VALUE DETERMINE THE TONE
0048  0332: 8D 07 A0             STA $A007      ;       OF THE OUTPUT (SPEED= 1000HZ)
0049  0335: 8D 05 A0             STA $A005      ;THIS STARTS TONE
0050  0338: A9 01                LDA #$1        ;TURN ON OUTPUT BIT-PB0
0051  033A: 8D 00 A0             STA $A000
0052  033D: 20 57 03             JSR DELAY      ;DELAYFOR ELEMENT TIME PERIOD
0053  0340: A9 00                LDA #$0
0054  0342: 8D 0B A0             STA $A00B      ;TURN OFF TONE
0055  0345: 8D 00 A0             STA $A000      ;TURN OFF OUTPUT BIT (PB0)
0056  0348: A0 01                LDY #$01
0057  034A: 20 57 03             JSR DELAY;     ;DELAY FOR 1 TIME PERIOD(SPACE BETWEEN ELEMENTS)
0058  034D: C6 F1                DEC COUNT      ;DECREMENT COUNT -SEE IF 8 BITS WERE ROTATED
0059  034F: D0 CA                BNE NEXT       ;  IF NOT, DO ANOTHER ELEMENT
0060  0351: A0 02           FINISH LDY #$2      ;DELAY FOR 3(TWO HERE PLUS PREVIOUS SPACE
0061  0353: 20 57 03             JSR  DELAY     ;   AT END OF LAST ELEMENT) TIME PERIODS(SPACE BET
0062  0356: 60                EXIT   RTS
0063                      ; THIS DELAYS FOR (Y REGISTER) *SPEED*.004 SECONDS
0064  0357: 98                DELAY  TYA
0065  0358: 0A                       ASL A
0066  0359: 0A                       ASL A
0067  035A: A8                       TAY
0068  035B: A5 F0           D3       LDA SPEED
0069  035D: A2 FA           D2       LDX #$FA
0070  035F: CA              D1       DEX
0071  0360: D0 FD                    BNE D1
0072  0362: 38                       SEC
0073  0363: E9 01                    SBC #$1
0074  0365: D0 F6                    BNE D2         ;DELAY FOR 7 TIME PERIODS
0075  0367: 88                       DEY            ;   (SPACE BETWEEN WORDS)
0076  0368: D0 F1                    BNE D3         ;RETURN FROM MORSE PROGRAM
0077  036A: 60                       RTS
0077  036B: A0 07           SPACE    LDY #$7
0077  036D: 20 57 03                 JSR DELAY
0077  0370: 60                       RTS
0077  0371: 73              TABLE  .BYTE $73,$31,$55,$32,$3F,$2F
0077  0372: 31
0078  0373: 6A
0078  0374: 32
0078  0375: 3F
0078  0376: 2F
0078  0377: 27                   BYTE $27,$23,$21,$20,$30,$38
0078  0378: 23
0079  0379: 21
0079  037A: 20
0079  037B: 30
0079  037C: 38
0079  037D: 3C                 .BYTE $3C,$3E,$01,$01,$01,$01
0079  037E: 3E
0080  037F: 01
0080  0380: 01
0080  0381: 01
0080  0382: 01
0080  0383: 01                 .BYTE $01,$4C,$01,$05,$1B,$1A
0080  0384: 4C
0081  0385: 01
```

Fig. 4-31 : Le programme de Morse (listing en grand dans l'appendice C)

Le son doit être produit pendant une durée déterminée, appelée « DELAY » (1). Cette durée peut être obtenue par des techniques hardware ou software. Nous utiliserons ici une boucle software. Le son doit être arrêté, en mettant à 0 le bit ACR7, lorsque le délai voulu est épuisé.

Le lecteur se reportera à l'ordinogramme de la figure 4-28, et il s'assurera qu'il comprend la séquence d'actions dont la mise en œuvre conditionne l'utilisation du temporisateur.

Le programme de Morse

On suivra ici l'ordinogramme de la figure 4-19, et développera le programme correspondant. Ce programme fera appel à un certain nombre de techniques spécifiques :

L'adressage indexé sera utilisé pour retrouver la forme binaire du code Morse d'un caractère ASCII donné.

Le temporisateur hardware sera utilisé pour générer un son de fréquence fixe. Un délai software réglera la durée du son.

```
0081  0386: 05
0081  0387: 18
0081  0388: 1A
0081  0389: 0C              .BYTE $0C,$02,$12,$0E,$10,$04
0081  038A: 02
0082  038B: 12
0082  038C: 0E
0082  038D: 10
0082  038E: 04
0082  038F: 17              .BYTE $17,$0D,$14,$07,$06,$0F
0082  0390: 0D
0083  0391: 14
0083  0392: 07
0083  0393: 06
0083  0394: 0F
0083  0395: 16              .BYTE $16,$1D,$0A,$08,$03,$09
0083  0396: 1D
0084  0397: 0A
0084  0398: 08
0084  0399: 03
0084  039A: 09
0084  039B: 11              .BYTE $11,$0B,$19,$1B,$1C
0085  039C: 0B
0085  039D: 19
0085  039E: 1B
0085  039F: 1C

SYMBOL TABLE:
 SPEED       00F0     COUNT       00F1     CHAR        00F2
 MORSE       0300     STARTB      0314     NEXT        031B
 SEND        0326     FINISH      0351     EXIT        0356
 DELAY       0357     D3          035B     D2          035D
 D1          035F     SPACE       036B     TABLE       0371
```

Fig. 4-31 : Le programme de Morse (suite)

(1) Pour les programmes dont le listing est obtenu par sortie directe de l'ordinateur, nous garderons les noms de variables anglais (N.d.T.)

Des boucles imbriquées permettront de multiplier la durée du délai.

Examinons maintenant le programme, ce qui suppose que l'accumulateur ait été préalablement chargé avec la valeur ASCII du caractère qu'on veut transmettre en Morse (cf. carte mémoire fig. 4-18). La vitesse de transmission, exprimée en millisecondes, est ajustable, permettant ainsi d'obtenir une plus grande souplesse. La variable SPEED, d'adresse 00F0, doit être initialisée avant d'entrer dans le programme. Par exemple, si SPEED reçoit la valeur 1 000, la durée d'un "." sera 1 000 × 0,001 = 1 seconde. Le programme se trouve en page 3. Il commence à l'adresse 0300, en hexadécimal.

Son début est le suivant :

```
SPEED = $00F0
COUNT = $00F1
CHAR = $00F2
* = $0300
```

Les quatre premières lignes sont des directives de l'assembleur. Les trois premières assignent, respectivement, les adresses 00F0, 00F1, 00F2 aux variables SPEED, COUNT et CHAR. La quatrième ordonne que la valeur du pseudo-compteur d'adresses soit 0300 hexadécimal. En d'autres termes, elle demande que la première instruction exécutable du programme réside à l'adresse-mémoire 0300.

Il convient d'abord de vérifier que le caractère présent dans l'accumulateur a un code autorisé.

```
MORSE   CMP  #$20   EST-CE UN ESPACE ?
        BEQ  SPACE
        CMP  #$2C   ERREUR SI < 2C
        BCC  EXIT
        CMP  #$5B   OU ⩾5B
        BCS  EXIT
```

Les deux premières lignes vérifient que le caractère contenu dans l'accumulateur est un « espace » (20 hexa). Si tel est bien le cas, on observe un délai de sept minutes, suivi du délai normal entre deux caractères.

Les quatre instructions suivantes permettent de s'assurer que le code ASCII est bien compris entre "2C" et "5A", bornes incluses. Telle est la gamme des caractères ASCII valides pour une

transmission Morse. En présence d'un caractère fautif, on détectera l'erreur, et sautera à l'adresse « EXIT ». Pour garder au programme son caractère simple et éducatif, aucune disposition particulière n'est prévue en EXIT, pour signaler l'erreur. Nous vous conseillons vivement (à titre d'exercice) d'ajouter en EXIT des instructions spécifiques signalant le caractère erroné trouvé dans l'accumulateur. Dans ce programme, il n'y aura tout simplement pas de transmission du caractère erroné.

Une fois qu'un caractère ASCII correct a été trouvé dans l'accumulateur, il doit être converti dans le code binaire qui sera utilisé pour générer la suite de sons recherchée. Le code Morse binaire, correspondant à chaque caractère ASCII autorisé, est stocké à la fin du programme de l'adresse 36 B à l'adresse 399. Nous recherchons le premier élément de la table, pour le code ASCII : 2C, le suivant pour le même code : (2C + 1), etc. C'est là un cas typique de l'avantage que présente *l'adressage indexé.* Mais un problème supplémentaire se présente ici : les caractères ASCII commencent à 2C, et non à 0 ou 1. La solution, très simple, apparaît ci-dessous :

```
TAX
LDA        TABLE-$2C, X
```

Le code ASCII est transféré dans le registre X, afin de servir de déplacement. Pour tenir compte du fait que la numérotation des caractères commence à 2C, la base de la table spécifiée n'est pas la base exacte (adresse 36 B), mais l'adresse moins 2C hexa. Le code Morse binaire est alors chargé dans l'accumulateur avec un seul accès mémoire indexé (cf. fig. 4-32).

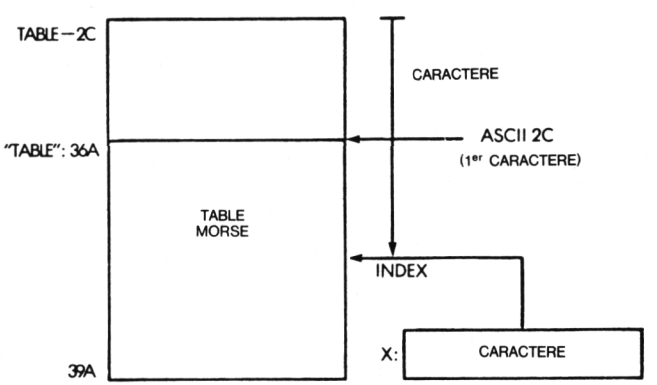


Fig. 4-32 : Obtention du code Morse à l'aide d'adressage indexé

Notre code Morse est maintenant dans l'accumulateur. Rappelons que ce code contient un 1 comme bit de DÉPART, suivi par les zéros et les uns qui représentent les points et les traits. Tous les bits inutilisés à gauche du bit de DÉPART sont à 0. On décalera à gauche le contenu de l'accumulateur, jusqu'au bit de départ, puis on utilisera les bits « effectifs » correspondant aux "." et aux " – " pour générer les sons. Voici le programme :

```
        LDY  #$08    NOMBRE DE BITS A DÉCALER
        STY  COUNT
STARTB  ASL  A
        DEC  COUNT
        BCC  STARTB  DÉCALE JUSQU'AU BIT
                     DE DÉPART
        STA  CHAR
NEXT    LDA  CHAR
        ASL  A       DÉCALE LE CODE MORSE
                     (1 = TRAIT, 0 = POINT)
        STA  CHAR    POINT = 1 UNITÉ DE TEMPS
        LDY  #$01    POINT PAR DÉFAUT
        BCC  SEND    SI RETENUE A 0, POINT
        LDY  #$03    SINON TRAIT
                     (3 UNITÉS DE TEMPS)
```

Le registre index Y devrait normalement servir de compteur pour provoquer l'arrêt des décalages de A après 8. Mais la routine SEND (émission), qui génère les sons, en a, elle aussi, usage : Y contient la durée du son à générer. La première idée qui vient alors à l'esprit est de réutiliser le registre index X, maintenant disponible. Malheureusement, X est, par convention, monopolisé par la routine DELAY. Aucun des deux registres index n'étant disponible comme compteur, on aura recours à une case-mémoire : la variable COUNT. Remarque importante : cette partie du programme aurait pu être codée avant que ne soient écrites les routines SEND et DELAY. Dans cette hypothèse, nous aurions probablement utilisé les registres index X ou Y pour conserver le nombre de bits à décaler de l'accumulateur. Nous ne nous serions aperçus que plus tard de la nécessité d'utiliser les mêmes registres dans les routines SEND et DELAY. La discipline de programmation prend ici toute son importance. Découvre-t-on que d'autres routines nécessitent

l'utilisation de X et Y ? Il faut alors revenir en arrière, et modifier le programme qui précède afin d'utiliser une case-mémoire, appelée COUNT, au lieu d'un registre. Hélas ! l'oubli de cette modification est une erreur classique. Conséquence : les autres routines vont détruire accidentellement le contenu de X ou Y, provoquant une sérieuse erreur de programmation. *Il est donc fortement recommandé d'avoir la discipline de préciser explicitement, dans les commentaires de début de routine, quels sont les registres changés ou détruits par elle.* Les conventions pour communiquer et passer des informations entre sous-programmes, ou segments d'un programme, doivent être parfaitement claires avant d'entreprendre l'écriture d'une nouvelle routine.

Les zéros placés le plus à gauche de l'accumulateur sont ignorés, et ce dernier est décalé jusqu'à un bit de DEPART. A partir de ce moment, chaque bit sortant par la gauche de l'accumulateur représente soit un "." soit un " – ", selon qu'il s'agit d'un 0 ou d'un 1. Une fois le bit sortant identifié, on se rend à l'adresse SEND pour générer le son approprié. Le contenu de l'accumulateur devant être changé par le traitement ultérieur, il doit être préservé en mémoire avant d'aller à SEND. C'est le but de la 2e instruction : STA CHAR.

	ADDRESS	WRITE	READ
TIMER 1	- - 04	T1L-L	T1C-L/ + clear T1 int flag
	- - 05	T1L-H + T1C-H + T1 L-L ►T1C-L + clear T1 int flag	T1C-H
	- - 06	T1L-L	T1L-L
	- - 07	T1L-H + clear T1 int flag	T1L-H
TIMER 2	- - 08	T2L-L	T2C-C + clear T2 int flag
	- - 09	T2C-H T2L-L ►T2CL + clear T2 int flag	T2C-H

Fig. 4-33 : Carte mémoire pour le temporisateur 1

Ayant ainsi préservé le contenu de l'accumulateur dans la case-mémoire CHAR, on chargera le registre index Y avec la durée correspondant au bit qui vient de sortir de l'accumulateur : 1 si c'est un point, 3 si c'est un trait. Le but du STA CHAR, suivi de LDA CHAR, peut n'être pas évident. Il répond à notre désir de rentrer dans ce programme à NEXT avec une instruction LDA CHAR.

La routine SEND (émission)

La routine SEND fait appel au temporisateur T1 du 6522 pour générer le son de fréquence convenue. La carte des adresses utilisées par ce temporisateur apparaît figure 4-33. Le temporisateur doit d'abord être mis en mode oscillateur. C'est ce que font :

```
SEND    LDA   #$C0
        STA   $A00B
```

On dépose la valeur C0 à l'adresse A00B, qui est celle de ACR (registre de commande auxiliaire). Elle met à 1 les bits 6 et 7, ce que requiert le temporisateur (cf. fig. 4-29 pour les détails). On dépose ensuite la valeur 0400 hexadécimal aux adresses A006, A007 :

```
        LDA   #$00
        STA   $A006
        LDA   #$04
        STA   $A007
```

Les adresses correspondent, respectivement, aux parties basse et haute de T1L (tampon), ce qui établit la fréquence du son à générer : 0400 en hexadécimal représente 1 024. Une demi-période du créneau équivaut approximativement à N + 2, ou 1 026. La période est donc T = 2 052 microsecondes, la fréquence devenant alors N = 1/T. Soit approximativement 500 Hz.

Il convient maintenant de démarrer le son, et de l'arrêter après la durée voulue. Le son est démarré par :

```
        STA   $A005
```

Cette instruction transfère le contenu du tampon dans le compteur, et déclenche le signal à générer, à l'extérieur. Nous

avons indiqué que ce programme met aussi, manuellement, PB0 à 1, de sorte que l'on puisse activer un appareil extérieur, tel qu'un émetteur, simultanément avec la génération du son dans le haut-parleur. On obtient ce résultat par :

```
LDA   #$01
STA   $A000
```

Nous supposerons que PB0 a été configurée en sortie, avant l'exécution du programme.

La durée du son est obtenue par le sous-programme DELAY : JSR DELAY. Nous l'examinons ci-dessous. Une fois cette durée écoulée, la coupure du son est opérée par :

```
LDA   #$00
STA   $A00B      EXTINCTION DU SON
STA   $A000      RAZ BIT DE SORTIE (PB0)
```

Il importe de respecter une unité de temps de silence entre deux sons. On parvient à ce résultat par :

```
LDY   #$01
JSR   DELAY      DÉLAI D'1 PÉRIODE
```

Il est, finalement, nécessaire de décrémenter notre compteur de bits, contenu dans la case-mémoire COUNT, afin de déterminer si d'autres bits doivent être décalés de l'accumulateur. On parvient à ce résultat par :

```
DEC   COUNT      8 BITS FAITS ?
BNE   NEXT       SINON RETOURNER
```

Lorsqu'un caractère complet a été transmis, il faut encore respecter un délai de 2 unités, pour séparer ce caractère du suivant. On parvient à ce résultat par :

```
FINISH   LDY   #$02
         JSR   DELAY
EXIT     RTS
```

Le sous-programme DELAY

Ce sous-programme marque un délai de : (contenu de Y) × (SPEED) × 0,001 seconde. Le délai sera donc le produit de 3 nombres. Nous utiliserons ici une technique de boucles imbriquées, pour éviter d'effectuer une multiplication classique. La routine apparaît ci-dessous :

```
DELAY   LDA   SPEED
D2      LDX   #$C6
D1      DEX
        BNE   D1
        SEC
        SBC   #$01
        BNE   D2
        DEY
        BNE   DELAY
        RTS
```

L'ordinogramme correspondant apparaît figure 4-34.

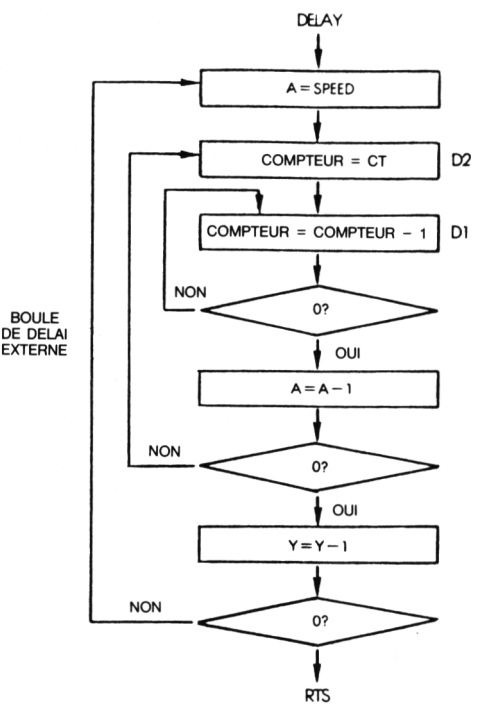


Fig. 4-34 : Ordinogramme de DELAY

La première boucle de délai correspond à D1. Calculons sa durée :

```
(3) LDA   SPEED
(2) LDX   #$C6        C6 HEX = 198 DEC
(2) DEX
(3) BNE   D1
```

La durée du délai introduit par les quatre premières instructions du sous-programme est : 3 + 2 + (2 + 3) × 198 − 1 = 994 microsecondes.

Les deux instructions suivantes sont :

```
(2) SEC
(2) SBC   #$01
```

Leur durée est de 2 μs chacune. Le délai additionnel est de 4 μs. Les deux instructions servent à soustraire 1 de l'accumulateur, parce que les deux registres index X et Y sont déjà utilisés, comme compteurs, par le programme. Il faut donc utiliser l'accumulateur comme troisième compteur. Malheureusement, il n'existe pas d'instruction de décrémentation qui opère directement sur l'accumulateur : on est contraint d'avoir recours à une soustraction en bonne et due forme. Le lecteur se rappellera que la retenue à 1 doit être forcée avant une soustraction. Tel est l'objet de l'instruction SEC, précédant SBC. L'instruction suivante est :

```
(2/3) BNE   D2
```

Il s'agit d'une seconde boucle de délai. Quand le branchement se produit, elle nécessite 3 μs, et 2 μs quand il ne se produit pas. Le temps écoulé depuis le point d'entrée DELAY jusqu'à ce point du programme est donc : 995 + 7 = 1 002 microsecondes = 1 milliseconde.

On génère un délai de 1 milliseconde chaque fois qu'on exécute la boucle D2. Comme cette dernière contient la valeur de SPEED, les deux boucles marquent un délai de : SPEED × 0,001 seconde. Tel était bien notre objectif. Lorsque ce délai de

SPEED × 0,001 seconde est terminé,

on exécute une boucle supplémentaire, qui utilise le registre Y :

```
DEY
BNE   DELAY
RTS
```

Cette dernière boucle multiplie le délai précédent par la valeur contenue dans Y. A ce point, nous avons obtenu le délai total désiré Y × SPEED × 0,001 seconde. Nous retournons alors (RTS).

Utilisation du programme

Pour utiliser ce programme, nous vous suggérons de choisir, au début, une vitesse de transmission lente, sauf si le Morse vous est familier, et de ne générer qu'un caractère à la fois. Lorsque votre programme fonctionnera correctement, vous devrez écrire un court sous-programme, fournissant des caractères à votre programme de Morse. Vous pourrez alors vérifier que la transmission Morse fonctionne bien pour n'importe quelle chaîne de caractères.

Exercice 4-2 : *Ecrire un sous-programme fournissant à votre programme une chaîne de N caractères, contenue dans une table commençant à l'adresse TABLE.*

Exercice 4-3 : *Lisez les caractères sur le clavier, et générez les signaux Morse correspondants.*

HORLOGE 24 HEURES

Nous développerons maintenant une routine d'horloge, qui entretiendra l'heure (en heures, minutes et secondes) dans trois emplacements mémoire consacrés. Si on le désire, ce programme peut très facilement être étendu, de telle sorte qu'il conserve, aussi, les fractions de seconde, ou toute autre unité. La carte d'implantation mémoire de ce programme est fournie par la figure 4-35. Comme d'habitude, nous avons réservé des emplacements de la page zéro pour les variables. Les heures, minutes et secondes sont respectivement rangées aux adresses 00F4 (hexa), 00F5 et 00F6. Une case supplémentaire 00F7 sert à contenir la variable COUNT.

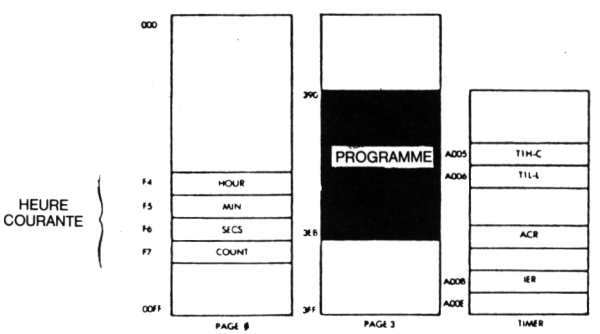


Fig. 4-35 : Carte d'implantation-mémoire du programme d'horloge

Pour démarrer l'horloge, il est nécessaire de taper, successivement, le programme, puis l'heure actuelle plus une minute, dans les emplacements SECS, MIN et HOUR.

Dans le cas du SYM, on entre A7 en A67E, et 03 en A67F. C'est un vecteur d'interruption, comme on le verra plus loin. On tape, enfin, G 0390. Au moment précis correspondant à l'heure indiquée, on tape CR (retour chariot).

A partir de ce moment, l'heure sera conservée dans SECS, MIN et HOUR.

La variable COUNT conserve des unités de 1/20 de seconde. Elle est initialisée à 20, puis décrémentée tous les 1/20 de seconde. Le signal de décrémentation est une interruption hardware, générée par un temporisateur du 6522. L'ordinogramme du programme apparaît figure 4-36. La première phase est une phase d'initialisation : le temporisateur est chargé avec la valeur appropriée pour générer une interruption, après 50 millisecondes (1/20 de seconde). La variable COUNT est initialisée à la valeur 20. On déclenche alors le temporisateur.

Lorsque le temporisateur arrive au time-out, 1/20 de seconde s'est écoulé : une interruption se produit. Alerté, le microprocesseur préserve ses registres, et recharge le compteur du temporisateur avec la valeur déterminée pour générer une autre interruption, 50 ms plus tard, et redémarrer le temporisateur. La variable COUNT est décrémentée, puisque 1/20 de seconde s'est écoulé. On teste si la valeur de COUNT est à 0. Si elle s'y trouve, on remet COUNT à 20, et l'adresse-mémoire SECS (nombre de secondes) est augmentée de 1. Dans le cas contraire on sort de la routine.

Chaque fois que SECS est augmentée de 1, on teste si elle atteint la valeur 60, et si c'est le cas, il faut la remettre à 0, et incrémenter MIN (le nombre de minutes). De la même façon, on teste si MIN a atteint la valeur 60. Si c'est le cas, MIN est remis à 0, et le nombre d'heures (HOUR) est incrémenté. Si le nombre d'heures atteint 24, il est remis à 0. Ensuite, on sort de la routine. Le programme est dans l'état dormant, jusqu'à réception de la prochaine interruption. Pour afficher l'heure, il suffit d'examiner le contenu des adresses mémoire F4, F5 et F6. On pourrait aussi écrire une routine d'affichage automatique.

Le programme apparaît figure 4-37, et s'explique de lui-même. Le premier segment du programme est l'initialisation, INIT, qui met la variable COUNT à 20 (décimal) = 14 hexadécimal. Il charge, en outre, le temporisateur avec la constante susceptible de générer un délai de 50 millisecondes. La carte-mémoire du temporisateur apparaît figure 4-35. On utilise le temporisateur T1 du 6522. La table montrant les bits qui conditionnent ce dispositif

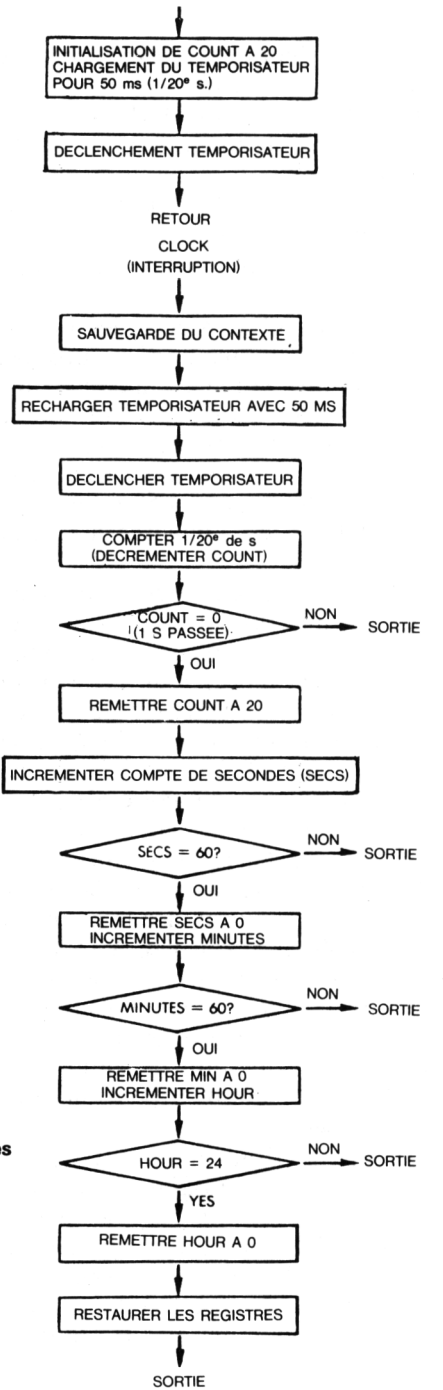


Fig. 4-36 : Horloge 24 heures

```
LINE #  LOC   CODE        LINE

                          ;FIRST LOAD A7 IN LOCATION A67E, AND 03 IN A67F
0002    0000              ;THIS IS A REAL TIME CLOCK ROUTINE WHICH MAINTAINS
0003    0000              ;THE CURRENT TIME IN THE LOCATIONS SEC (00F6), MIN
0004    0000              ;(00F5), AND HOUR (00F4) (24 HOUR TIME). IT IS BRANCHED TO
0005    0000              ;BY THE TIME OUT OF THE INTERRUPT TIMER, WHICH
0006    0000              ;CAUSES AN INTERRUPT AND BRANCH TO THE CLOCK
0007    0000              ;ROUTINE TWENTY TIMES PER SECOND. THE CLOCK ROUTINE
0008    0000              ;AND INTERVAL TIMER MUST BE INITIALIZED FIRST. THE
0009    0000              ;CODE 'INIT' DOES THIS, AND IT MUST BE BRANCHED TO TO
0010    0000              ;START THE CLOCK. TO INITIALIZE, PUT THE CURRENT TIME
0011    0000              ;THE CLOCK ROUTINE WILL BE STARTED IN SEC, MIN, AND
                          ;HOUR, THEN ISSUE THE COMMAND 'GO 0390 CR' AT THAT
                          ;EXACT TIME. NOTHING ELSE MUST BE DONE
0012    0000              COUNT = $00F7          ;COUNTER FOR TWENTIETHS OF A SEC
0013    0000              SECS = $00F6           ;CURRENT TIME
0014    0000              MIN = $00F5
0015    0000              HOUR = $00F4
0016    0000              ACR = $A00B            ;TIMER MODE REGISTER
0017    0000              TILL = $A006           ;LOW ORDER TIMER CONSTANT
0018    0000              TIHC = $A005           ;HIGH ORDER TIMER CONSTANT
0019    0000                    * = $0390
0020    0390  A9 14       INIT  LDA #$14               ;SET TO FIRST TWENTY
0021    0392  85 F7             STA COUNT              ;COUNTS
0022    0394  8D 0B A0          STA ACR                ;SET BITS 8 AND 7 LOW
                                                       ;IN ACR
0023    0397  A9 C0             LDA #$C0               ;SET BITS 8 AND 7 HIGH IN
0024    0399  8D 0E A0          STA $A00E              ;THE INTERRUPT ENABLE
                                                       ;REGISTER (TO ENABLE
                                                       ;INTERRUPTS FROM TIMER 1)
0025    039C  A9 50             LDA #$50               ;STORE C350 IN TIMER
0026    039E  8D 06 A0          STA TILL               ;  (DELAY CONSTANT FOR
0027    03A1  A9 C3             LDA #$C3               ;  50 MS)
0028    03A3  8D 05 A0          STA TIHC               ;THIS STARTS TIMER
0029    03A6  60                RTS                    ;RETURN TO MONITOR
0030    03A7  08          CLOCK PHP                    ;SAVE STATUS
0031    03A8  48                PHA
0032    03A9  F8                SED
0033    03AA  A9 50             LDA #$50               ;STORE C350 IN TIMER
0034    03AC  8D 06 A0          STA TILL               ;  (DELAY CONSTANT FOR
0035    03AF  A9 C3             LDA #$C3               ;  50 MS)
0036    03B1  8D 05 A0          STA TIHC               ;THIS STARTS TIMER
0037    03B4  C6 F7             DEC COUNT              ;DECREMENT COUNT OF
                                                       ;TWENTY
0038    03B6  D0 31             BNE EXIT               ;EXIT IF WE HAVE NOT
                                                       ;COUNTED TO TWENTY YET
0039    03B8  A9 14             LDA #$14               ;ELSE RESTORE COUNT--
0040    03BA  85 F7             STA COUNT              ;A FULL SECOND HAS PASSED
0041    03BC  A9 01             LDA #$01
0042    03BE  18                CLC
0043    03BF  65 F6             ADC SECS               ;ADD 1 TO SEC
0044    03C1  85 F6             STA SECS
0045    03C3  C9 60             CMP #$60               ;SEE IF 60 SECONDS
0046    03C5  D0 22             BNE EXIT               ;IF NOT, EXIT
0047    03C7  A9 00             LDA #$00               ;ELSE RESET SECONDS TO 0
0048    03C9  85 F6             STA SECS
0049    03CB  A9 01             LDA #$01
0050    03CD  18                CLC
0051    03CE  65 F5             ADC MIN                ;AND ADD 1 TO MINUTES
0052    03D0  85 F5             STA MIN
0053    03D2  C9 60             CMP #$60               ;SEE IF 60 MINUTES
0054    03D4  D0 13             BNE EXIT               ;IF NOT, EXIT
0055    03D6  A9 00             LDA #$00
0056    03D8  85 F5             STA MIN                ;ELSE RESET MINUTES TO 0
0057    03DA  A9 01             LDA #$01
0058    03DC  18                CLC
0059    03DD  65 F4             ADC HOUR               ;AND ADD 1 TO HOUR
0060    03DF  85 F4             STA HOUR
0061    03E1  C9 24             CMP #$24               ;SEE IF 24 HOURS
0062    03E3  D0 04             BNE EXIT               ;IF NOT, EXIT
0063    03E5  A9 00             LDA #$00
0064    03E7  85 F4             STA HOUR               ;ELSE RESET HOUR TO 0
0065    03E9  68          EXIT  PLA                    ;RESTORE STATUS
0066    03EA  28                PLP
0067    03EB  40                RTI

ERRORS = 0000 <0000>

SYMBOL TABLE

SYMBOL    VALUE

ACR       A00B     CLOCK   03A7     COUNT   00F7     EXIT    03E9
HOUR      00F4     INIT    0390     MIN     00F5     PLS     03EA
SECS      00F6     TIHC    A005     TILL    A006

END OF ASSEMBLY
```

Fig. 4-37 : Le programme d'horloge 24 heures (listing en grand à l'appendice C)

se trouve aux figures 4-25 et 4-29. Le temporisateur peut être utilisé en mode monostable, ou en mode oscillateur. Dans la première hypothèse, on obtient une seule interruption (et, éventuellement, une impulsion de sortie sur PB7), chaque fois que le compteur arrive à 0. Dans la seconde, le compteur est rechargé automatiquement, à partir du tampon interne : on obtient continuellement des interruptions (et éventuellement un signal carré sur PB7). La sortie sur PB7 n'étant pas, ici, utilisée, on mettra à 0 le bit 7 de ACR (registre de commande auxiliaire). Le choix se pose, alors, entre le mode monostable, et le mode oscillateur. En monostable, on doit explicitement recharger le compteur, lors de chaque interruption. En oscillateur, le temporisateur le recharge automatiquement, à partir de son tampon. Mais l'indicateur d'interruption doit être remis à 0, explicitement, soit en écrivant dans T1C-H, soit en agissant directement sur lui. Les deux options sont essentiellement équivalentes, du point de vue de l'effort de programmation. Le mode oscillateur est susceptible de donner une mesure du temps plus exacte, dans la mesure où le temporisateur tourne continuellement, et se déplace automatiquement, de 0 à la valeur correspondant au délai de 50 ms. Comme nous avons utilisé le mode oscillateur dans le programme de Morse, nous utiliserons ici le monostable. Toutefois, nous conseillerons au lecteur d'expérimenter l'autre mode, à titre d'exercice. Le mode monostable est spécifié en mettant le bit ACR6 à 0. Tous les autres bits sont inutilisés, et donc mis à 0. Le bit 7 est à 0 pour indiquer que PB7 est désactivée.

Il faut, ensuite, conditionner convenablement le registre d'activation des interruptions IER. Pour mettre à 1 des bits de IER, on doit mettre à 1 le bit 7. Pour chaque 1 spécifié dans les bits 0 à 6 du motif écrit, un 1 sera écrit dans le registre IER, activant la condition correspondante. Par contre, la spécification d'un 0 ne remet pas à 0 le bit de IER : il le laisse inchangé. Pour ce faire, il faut spécifier un 0 en position 7, et un 1 en chaque position annulable. Notre but est ici, tout simplement, d'autoriser l'interruption émanant de T1 (bit 6). Nous écrirons donc à l'adresse mémoire correspondant à IER la valeur : "11000000", c'est-à-dire « C0 », en hexadécimal (cf. chap. II pour des détails).

Dernière étape : le chargement de la constante appropriée dans le temporisateur, pour générer le délai qui créera une interruption, après 50 ms. La valeur C350 hexadécimal sera chargée dans le compteur. Dans la routine INIT la partie basse du tampon est chargée avant la partie haute du compteur. Le fait de charger la partie haute du tampon entraîne, automatiquement, le transfert de la partie basse du tampon dans la partie basse du compteur, ainsi que le démarrage du temporisateur.

Le sous-programme INIT apparaît ci-dessous :

```
COUNT  = $00F7      COMPTE DES 20e DE SECONDE
SECS   = $00F6
MIN    = $00F5
HOUR   = $00F4
ACR    = $A00B      DÉTERMINE LE MODE
                    DU TEMPORISATEUR
T1LL   = $A006      TAMPON BAS DU TEMPORISATEUR
T1CH   = $A005      COMPTEUR HAUT DU TEMPORI-
                    SATEUR

INIT   LDA  #$14    PREMIER COMPTE DE 20
STA    STA  COUNT
       STA  ACR     BITS 7 ET 6 DE ACR A 0
       LDA  #$C0    BITS 7 ET 6 A 1
       STA  $A00E   DANS LE REGISTRE ACTIVATION
                    INTERRUPTIONS
       LDA  #$50    ÉCRITURE DE C350 DANS LE TEM-
       STA  T1LL    PORISATEUR (CTE POUR 50 MS)
       LDA  #$C3
       STA  T1CH    DÉMARRE LE TEMPORISATEUR
       RTS
```

L'initialisation est maintenant réalisée. On exécutera le programme principal (routine d'interruption) à partir de l'adresse CLOCK. A noter : toutes les additions à la routine CLOCK sont effectuées en mode décimal, par l'intermédiaire de l'instruction SED. De cette manière, lorsqu'on affichera le contenu des mémoires SECS, MIN et HOUR, l'affichage se fera à raison d'un chiffre par LED, en décimal, et non en hexadécimal.

Au terme de l'exécution de INIT, nous retournerons au moniteur. Si aucune touche du clavier n'est manœuvrée, rien ne se passera, jusqu'à la première interruption de time-out. Dès la détection de cette interruption le branchement automatique sur CLOCK intervient. Quand une interruption se produit, le 6502 se branche automatiquement à l'adresse FFFE-FFFF, où il trouve le vecteur d'interruption. Autrement dit, la prochaine adresse qui devra être mise dans le compteur ordinal. Sur le SYM, l'utilisateur

charge au préalable les adresses A67E et A67F, avec le vecteur d'interruption désiré. Le moniteur du SYM, qui est en cours d'exécution dès lors qu'un programme utilisateur n'est lui-même pas en cours, opère automatiquement la duplication du contenu des adresses A600-A67F, aux adresses FF80 à FFFF.

Le contenu de A67E-A67F est donc copié automatiquement par le moniteur du SYM dans FFFE, FFFF. A l'arrivée de l'interruption, il suffit de se brancher à FFFE, FFFF, pour trouver les 16 bits à envoyer dans le compteur ordinal.

CLOCK est la routine d'interruption exécutée à chaque fois qu'une interruption est reçue. Elle sauve les registres P (registre d'état) et A (accumulateur), mais pas les autres, dont elle ne se sert pas.

Elle recharge ensuite le compteur avec la valeur C350 hexa = 50 000 décimal, et redémarre le temporisateur. Le chargement du compteur remet automatiquement à 0 l'interruption précédente.

La routine effectue alors une série de tests. Elle vérifie que la variable COUNT a bien atteint la valeur 20, que MIN et SECS ont, de leur côté, atteint la valeur 60, et HOUR la valeur 24. Si l'une de ces variables a, en effet, atteint sa valeur limite, elle est remise à 0, comme le montrent l'ordinogramme de la figure 4-36 et le programme de la figure 4-37.

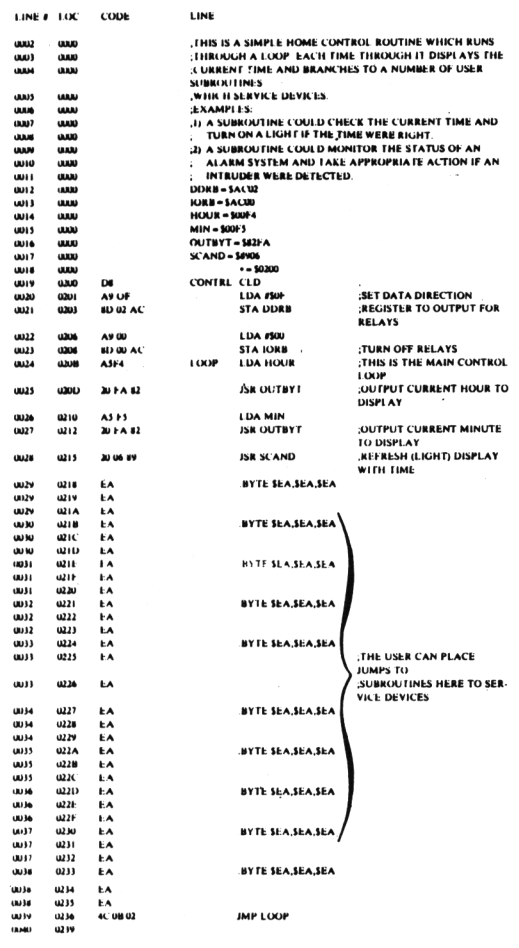
La routine se termine par un RTI (retour d'interruption), après restauration des deux registres A et P sauvegardés auparavant.

PROGRAMME DE CONTRÔLE DOMESTIQUE

Un programme de contrôle domestique généralisé surveille l'état d'une horloge 24 heures, et d'un système d'alarme. Il réalise différentes opérations en fonction de l'heure, ou de la condition d'alarme détectée. Nous utiliserons le programme d'horloge 24 heures développé ci-dessus. L'heure sera affichée, et conditionnera plusieurs actions spécifiques, réalisées en fermant un ou plusieurs relais. Le programme apparaît figure 4-38. Le registre direction du port B est mis à 0F hexadécimal, pour placer les quatre bits de droite en sortie (pour les relais). Seuls les bits connectés à des relais sont, évidemment, concernés. Il est préférable de maintenir les autres en entrée. Comme d'habitude, et à titre de précaution, une instruction explicite, destinée à mettre les relais à l'arrêt est incluse dans le programme. Comment ? En déposant la valeur 00 à l'adresse mémoire de IORB (adresse AC00).

On utilise, pour faciliter les sorties, deux routines, qui font partie du moniteur SYM.

L'accumulateur est chargé à partir de l'adresse HOUR, qui contient l'heure exprimée en heures (cf. la routine d'horloge 24 heures). On appelle ensuite le sous-programme OUTBYT, qui



```
LINE #  LOC   CODE        LINE

0002  0000                ;THIS IS A SIMPLE HOME CONTROL ROUTINE WHICH RUNS
0003  0000                ;THROUGH A LOOP EACH TIME THROUGH IT DISPLAYS THE
0004  0000                ;CURRENT TIME AND BRANCHES TO A NUMBER OF USER
                          SUBROUTINES
0005  0000                ;WHICH SERVICE DEVICES.
0006  0000                ;EXAMPLES:
0007  0000                ;1) A SUBROUTINE COULD CHECK THE CURRENT TIME AND
0008  0000                ;     TURN ON A LIGHT IF THE TIME WERE RIGHT.
0009  0000                ;2) A SUBROUTINE COULD MONITOR THE STATUS OF AN
0010  0000                ;     ALARM SYSTEM AND TAKE APPROPRIATE ACTION IF AN
0011  0000                ;     INTRUDER WERE DETECTED.
0012  0000                DDRB = $AC02
0013  0000                IORB = $AC00
0014  0000                HOUR = $00F4
0015  0000                MIN = $00F5
0016  0000                OUTBYT = $82FA
0017  0000                SCAND = $8906
0018  0000                        * = $0200
0019  0200  D8             CONTRL  CLD
0020  0201  A9 0F                  LDA #$0F          ;SET DATA DIRECTION
0021  0203  8D 02 AC               STA DDRB          ;REGISTER TO OUTPUT FOR
                                                     RELAYS
0022  0206  A9 00                  LDA #$00
0023  0208  8D 00 AC               STA IORB          ;TURN OFF RELAYS
0024  020B  A5F4           LOOP    LDA HOUR          ;THIS IS THE MAIN CONTROL
                                                     LOOP
0025  020D  20 FA 82               JSR OUTBYT        ;OUTPUT CURRENT HOUR TO
                                                     DISPLAY
0026  0210  A5 F5                  LDA MIN
0027  0212  20 FA 82               JSR OUTBYT        ;OUTPUT CURRENT MINUTE
                                                     TO DISPLAY
0028  0215  20 06 89               JSR SCAND         ;REFRESH (LIGHT) DISPLAY
                                                     WITH TIME
0029  0218  EA                     .BYTE $EA,$EA,$EA
0029  0219  EA
0029  021A  EA
0030  021B  EA                     .BYTE $EA,$EA,$EA
0030  021C  EA
0030  021D  EA
0031  021E  EA                     .BYTE $EA,$EA,$EA
0031  021F  EA
0031  0220  EA
0032  0221  EA                     .BYTE $EA,$EA,$EA
0032  0222  EA
0032  0223  EA
0033  0224  EA                     .BYTE $EA,$EA,$EA
0033  0225  EA                                       ;THE USER CAN PLACE
                                                     JUMPS TO
0033  0226  EA                                       ;SUBROUTINES HERE TO SER-
                                                     VICE DEVICES
0034  0227  EA                     .BYTE $EA,$EA,$EA
0034  0228  EA
0034  0229  EA
0035  022A  EA                     .BYTE $EA,$EA,$EA
0035  022B  EA
0035  022C  EA
0036  022D  EA                     .BYTE $EA,$EA,$EA
0036  022E  EA
0036  022F  EA
0037  0230  EA                     .BYTE $EA,$EA,$EA
0037  0231  EA
0037  0232  EA
0038  0233  EA                     .BYTE $EA,$EA,$EA
0038  0234  EA
0038  0235  EA
0039  0236  4C 0B 02               JMP LOOP
0040  0239

ERRORS = 0000/0000
```

Fig. 4-38 : Programme de contrôle domestique (listing en grand à l'appendice C)

```
SYMBOL TABLE
SYMBOL   VALUE

CONTRL   0200   DDRB   AC02   HOUR   00F4   IORB   AC00
LOOP     0208   MIN    00F5   OUTBYT 82FA   SCAND  8906

END OF ASSEMBLY
```

Fig. 4-38 : Programme de contrôle domestique (suite)

a pour effet d'afficher l'heure sur les LED de la carte. Les minutes sont de la même façon affichées en chargeant l'accumulateur à partir de la mémoire MIN, et en appelant OUTBYT.

La routine OUTBYT se trouve à l'adresse 82FA dans le moniteur. Elle affiche le contenu de A, sous la forme de deux chiffres hexadécimaux, avant d'appeler la routine SCAND du moniteur (à l'adresse 8906), et de balayer une fois l'afficheur. L'heure est affichée, et il s'agit maintenant d'exécuter une instruction de saut appropriée. Cette opération est soumise à une condition préétablie, qui peut varier avec chaque application, et sera donc laissée en blanc. Le soin de la remplir est laissé au lecteur. A titre d'exercice, nous vous suggérons de fermer les relais à deux ou trois instants spécifiés, séparés par quelques minutes. Le bruit provoqué par la fermeture des relais indique que le programme fonctionne correctement. Cette vérification est indispensable avant de brancher un quelconque appareil réel sur les relais.

COMPOSEUR DE NUMÉROS DE TÉLÉPHONE

Nous allons développer un programme capable de composer un numéro de téléphone dès le moment où il a été stocké en mémoire. Il suffit pour cela d'envoyer, avec un téléphone ordinaire (à cadran tournant) des impulsions sur la ligne. L'opération ne devrait pas présenter de difficultés.

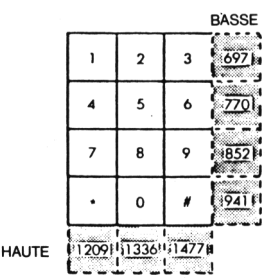


Fig. 4-39 : Les fréquences du téléphone

Le programme devra être capable de générer les fréquences sonores utilisées, aux Etats-Unis pour les téléphones à touches. La table des fréquences utilisées apparaît figure 4-39. Chaque chiffre provoque la génération de deux sons. Les différentes fréquences ont été choisies avec soin, par la compagnie du téléphone, pour éviter la possibilité d'harmoniques parasites, et pour permettre l'utilisation de la bande passante la plus étroite possible. Ces fréquences vont de 697 Hertz à 1 477 Hertz, comme l'indique la figure.

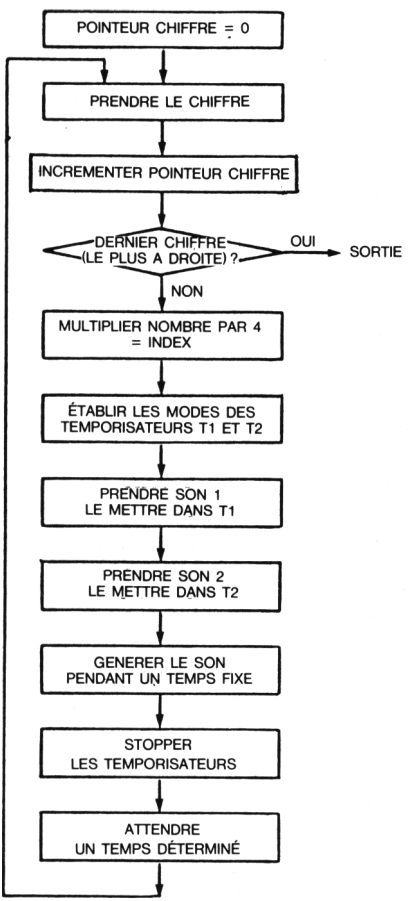


Fig. 4-40 : Ordinogramme du composeur de numéros de téléphone

Notre programme va générer simultanément deux sons, qui vont être envoyés sur le même haut-parleur. Les fréquences devront être exactes, pour être reconnaissables par l'équipement de commutation du téléphone. Nous ferons, pour cela, appel à deux temporisateurs T1 provenant de deux 6522 de notre carte micro-ordinateur. Chaque temporisateur générera une fréquence. La sortie des deux temporisateurs sera envoyée sur le haut-parleur. Pour augmenter la sûreté des résultats, il est vivement recommandé d'attaquer le haut-parleur par un amplificateur opérationnel. Cela ne change toutefois rien au programme. L'ordinogramme apparaît figure 4-40. Le nombre de chiffres du numéro à composer est indifférent. Le programme peut traiter un numéro, quelle que soit sa longueur. Le premier chiffre à « composer » est obtenu en mémoire. Une table d'équivalence, indiquant les périodes de sons à générer pour chaque chiffre, est conservée en mémoire. Précisons bien que c'est la *demi-période* qui est tabulée.

Comme deux sons sont associés à chaque chiffre, la table comporte quatre octets pour chaque chiffre. La valeur du chiffre doit être multipliée par quatre pour y servir d'index.

Les deux valeurs de la table seront extraites, et envoyées, respectivement, dans les temporisateurs A et B, qui vont alors démarrer. Les deux sons seront produits automatiquement, pendant une durée déterminée (disons, une demi-seconde ou une seconde). On forcera ensuite un intervalle de silence, et on cherchera le chiffre suivant en mémoire. Le processus doit être répété jusqu'à ce que tous les chiffres aient été composés. L'ordinogramme est évident. Examinons maintenant le programme (fig. 4-41).

```
LINE #  LOC   CODE      LINE

0002    0000            ;THIS IS A PROGRAM WHICH DIALS PRE STORED
0003    0000            ;TELEPHONE NUMBERS. IT PRODUCES A TWO TONE OUTPUT
0004    0000            ;THROUGH A SPEAKER HOOKED UP IN CONFIGURATION 2
0005    0000            ;(TWO TONES--SEE SPEAKER). THESE TONES WILL ACTIVATE
0006    0000            ;A STANDARD TOUCH TONE PHONE WHEN THE SPEAKER IS
0007    0000            ;PLACED DIRECTLY OVER THE MOUTH PIECE OF THE TELE-
0008    0000            ;PHONE. TO USE THE PROGRAM, PLACE THE PHONE
0009    0000            ;NUMBER(S) ANYWHERE IN MEMORY, ONE DIGIT PER BYTE,
0010    0000            ;AND ENDING WITH 0F (HEX). FOR EXAMPLE, THE NUMBER
0011    0000            ;555-1212 WOULD BE 05 05 05 01 02 01 02 0F (ALL HEX) IN
0012    0000            ;MEMORY. THEN PLACE THE ADDRESS OF THE NUMBER,
0013    0000            ;LOW BYTE FIRST, IN THE LOCATIONS 00C0 AND 00C1.
0014    0000            ;THEN EITHER GO TO THIS ROUTINE FROM THE MONITOR
                        ;OR JSR TO IT FROM ANOTHER PROGRAM.
0015    0000            NUMPTR = $00C0      ;THIS POINTS TO THE ADDRESS OF
                                            ;THE TELEPHONE NUMBER
0016    0000            ONDEL = $40         ;THIS IS THE DELAY CONSTANT FOR
                                            ;THE TIME WHEN THE
0017    0000            OFFDEL = $20        ;DELAY CONSTANT FOR THE TIME
                                            ;WHEN THE TONES ARE 0
0018    0000            DELCON = $FF        ;GENERAL PURPOSE DELAY
                                            ;CONSTANT
0019    0000            ACR1 = $A00B        ;THESE ARE THE TIMER MODE
                                            ;REGISTERS (TIMER 1)
0020    0000            ACR2 = $A00B        ;(TIMER 2)
0021    0000            T1CH = $A005        ;THIS IS THE TIMER 1 COUNTER
                                            ;(HIGH BYTE)
0022    0000            T1LH = $A007        ;TIMER 1 LATCH (HIGH BYTE)
0023    0000            T1LL = $A004        ;  (LOW BYTE)
0024    0000            T2CH = $A005        ;SAME AS TIMER 1 -- FOR TIMER 2
0025    0000            T2LH = $A007
0026    0000            T2LL = $A004
0027    0000                    *=$0300
0028    0300  A0 00     PHONE   LDY #$00             ;INDEX FOR DIGITS OF
                                                     ;PHONE NUMBER
0029    0302  B1 C0     DIGIT   LDA (NUMPTR),Y       ;GET DIGIT
0030    0304  C8                INY
0031    0305  C9 0F             CMP #$0F             ;SEE IF END OF PHONE
                                                     ;NUMBER
```

Fig. 4-41 : Programme composeur de numéros de téléphone (listing en grand à l'appendice C)

```
0032  0307  D0 01          BNE NOEND
0033  0309  60             RTS             ;RETURN IS SO (TO
                                           ;MONITOR OR CALLING
                                           ;PROGRAM)
0034  030A  0A EA EA  NOEND  ASL A          ;MULTIPLY NUMBER BY
                                           ;FOUR TO INDEX TABLE
0035  030D  0A EA EA        ASL A          ;  (EACH TABLE ENTRY IS
                                           ;  4 BYTES)
0036  0310  AA             TAX             ;X = INDEX FOR TABLE
0037  0311  A9 CD          LDA #$C0
0038  0313  8D 0B A0       STA ACR1        ;SET TIMER MODE TO FREE
                                           ;RUNNING ON BOTH TIMERS
0039  0316  8D 0B AC       STA ACR2
0040  0319  BD 5D 03       LDA TABLE,X     ;GET LOW ORDER, FIRST
                                           ;TONE
0041  031C  8D 04 A0       STA T1LL        ;STORE IN TIMER 1
0042  031F  E8             INX
0043  0320  BD 5D 03       LDA TABLE,X     ;GET HIGH ORDER, FIRST
                                           ;TONE
0044  0323  8D 07 A0       STA T1LH        ;STORE TIMER 1
0045  0326  8D 05 A0       STA T1CH        ;THIS STARTS TIMER 1
                                           ;GOING
0046  0329  E8             INX
0047  032A  BD 5D 03       LDA TABLE,X     ;GET LOW ORDER, SECOND
                                           ;TONE
0048  032D  8D 04 AC       STA T2LL        ;STORE IN TIMER 2
0049  0330  E8             INX
0050  0331  BD 5D 03       LDA TABLE,X     ;GET HIGH ORDER, SECOND
                                           ;TONE
0051  0334  8D 07 AC       STA T2LH        ;STORE IN TIMER 2
0052  0337  8D 05 AC       STA T2CH        ;THIS STARTS TIMER 2
                                           ;GOING
0053  033A  A2 40          LDX #ONDEL      ;GET TONES-ON DELAY
                                           ;CONSTANT
0054  033C  20 55 03  ON    JSR DELAY       ;DELAY WHILE TONE IS ON
0055  033F  CA             DEX
0056  0340  D0 FA          BNE ON
0057  0342  A9 00          LDA #$00
0058  0344  8D 0B A0       STA ACR1        ;TURN BOTH TIMERS OFF
0059  0347  8D 0B AC       STA ACR2
0060  034A  A2 20          LDX #OFFDEL     ;GET TONES-OFF DELAY
                                           ;CONSTANT
0061  034C  20 55 03  OFF   JSR DELAY       ;DELAY WHILE TONE IS OFF
0062  034F  CA             DEX
0063  0350  D0 FA          BNE OFF
0064  0352  4C 02 03       JMP DIGIT       ;GO BACK FOR NEXT DIGIT
                                           ;OF PHONE NUMBER
0065  0355                ;
0066  0355                ;THIS IS A SIMPLE DELAY ROUTINE FOR THE TONE ON AND
                          ;OFF PERI
0067  0355                ;
0068  0355  A9 FF    DELAY  LDA #DELCON     ;GET DELAY CONSTANT
0069  0357  38        WAIT   SEC             ;DELAY FOR THAT LONG
0070  0358  E9 01          SBC #$01
0071  035A  D0 FB          BNE WAIT
0072  035C  60             RTS
0073  035D                ;
0074  035D                ;THIS IS A TABLE OF THE CONSTANTS FOR THE TONE
0075  035D                ;FREQUENCIES FOR EACH TELEPHONE DIGIT. THE
0076  035D                ;CONSTANTS ARE TWO BYTES LONG, LOW BYTE FIRST.
0077  035D                ;
0078  035D  13    TABLE    .BYTE $13,$02,$76,$01   ;TWO TONES FOR '0'
0078  035E  02
0078  035F  76
0078  0360  01
0079  0361  CD             .BYTE $CD,$02,$9E,$01   ;TWO TONES FOR '1'
0079  0362  02
0079  0363  9E
0079  0364  01
0080  0365  CD             .BYTE $CD,$02,$76,$01   ;   '2'
0080  0366  02
0080  0367  76
0080  0368  01
0081  0369  CD             .BYTE $CD,$02,$53,$01   ;   '3'
0081  036A  02
0081  036B  53
0081  036C  01
0082  036D  89             .BYTE $89,$02,$9E,$01   ;   '4'
0082  036E  02
0082  036F  9E
0082  0370  01
0083  0371  89             .BYTE $89,$02,$76,$01   ;   '5'
0083  0372  02
0083  0373  76
0083  0374  01
0084  0375  89             .BYTE $89,$02,$53,$01   ;   '6'
0084  0376  02
0084  0377  53
0084  0378  01
0085  0379  4B             .BYTE $4B,$02,$9E,$01   ;   '7'
0085  037A  02
0085  037B  9E
0085  037C  01
0086  037D  4B             .BYTE $4B,$02,$76,$01   ;   '8'
0086  037F  02
0086  037E  76
0086  0380  01
0087  0381  4B             .BYTE $4B,$02,$53,$01   ;   '9'
```

Fig. 4-41 : Programme composeur de numéros de téléphone (suite)

```
0087   0382   02
0087   0383   53
0087   0384   01
0088   0385                    .END

ERRORS = 0000 <0000>

SYMBOL TABLE

SYMBOL   VALUE

ACR1     A00B    ACR2     AC0B    DELAY    0355    DELCON   00FF
DIGIT    0302    NOEND    030A    NUMPTR   00C0    OFF      034C
OFFDEL   0020    ON       033C    ONDEL    0040    PHONE    0300
T1CH     A005    T1LH     A007    T1LL     A004    T2CH     AC05
T2LH     AC07    T2LL     AC04    TABLE    035D    WAIT     0357

END OF ASSEMBLY
```

Fig. 4-41 : Programme composeur de numéros de téléphone (suite)

Le registre Y sert de pointeur, pour désigner le chiffre courant du numéro de téléphone que l'on est en train de composer. Il est tout d'abord initialisé à 0 :

```
PHONE     LDY    #$00
```

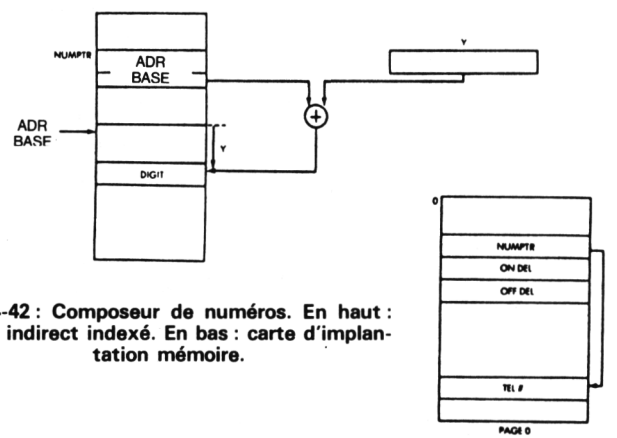


Fig. 4-42 : Composeur de numéros. En haut : accès indirect indexé. En bas : carte d'implantation mémoire.

On recherchera le chiffre à l'aide d'une technique d'adressage indirect indexé (cf. fig. 4-42). Le numéro de téléphone complet est supposé avoir été rangé séquentiellement à partir de l'adresse contenue dans "NUMPTR", et terminé par la valeur "0F", qui sert de marque de fin.

```
DIGIT     LDA    (NUMPTR), Y    PRENDRE LE CHIFFRE
```

On incrémentera le registre Y pour qu'à chaque fois il pointe vers le chiffre suivant. On testera qu'on est bien arrivé au dernier chiffre "0F", ce qui marquera la fin du programme :

```
        INY
        CMP   #$0F
        BNE   NOEND
        RTS
```

Supposons que le dernier chiffre du numéro n'ait pas encore été atteint. Dans ce cas, il faudra multiplier la valeur du chiffre par quatre, puisque nous avons déjà mis en évidence que la table d'équivalence chiffres-périodes contient quatre octets pour chaque chiffre. On effectuera la multiplication par quatre, à l'aide de deux décalages à gauche successifs. Le résultat sera rangé dans le registre X, de telle sorte qu'il puisse servir d'index :

```
NOEND   ASL   A
        ASL   A
        TAX
```

Les deux temporisateurs seront mis en mode oscillateur, avec sortie sur PB7 :

```
        LDA   #$C0
        STA   ACR1
        STA   ACR2
```

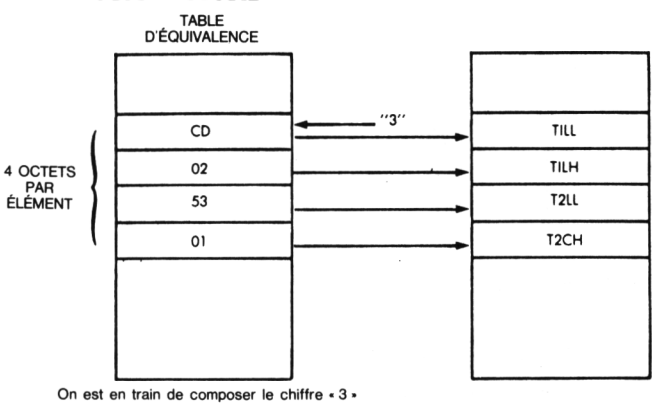


Fig. 4-43 : Chargement du temporisateur

Les temporisateurs seront chargés avec, chacun, la demi-période récupérée dans la table d'équivalence (cf. fig. 4-43) :

```
        LDA   TABLE, X
        STA   T1LL
        INX
```

```
        LDA   TABLE, X
        STA   T1LH
        STA   T1CH
        INX
        LDA   TABLE, X
        STA   T2LL
        INX
        LDA   TABLE, X
        STA   T2LH
        STA   T2CH
```

Une fois les deux temporisateurs activés, il suffit de maintenir le son pendant une durée déterminée, spécifiée ici par la constante ONDEL. Le délai est obtenu par le sous-programme DELAY, et une boucle auxiliaire "ON" :

```
        LDX   #ONDEL
ON      JSR   DELAY
        DEX
        BNE   ON
```

On arrête enfin les deux temporisateurs :

```
        LDA   #$00
        STA   ACR1
        STA   ACR2
```

Avant de générer un temps de silence :

```
        LDX   #OFFDEL
OFF     JSR   DELAY
        DEX
        BNE   OFF
```

Le programme revient au début, pour générer les sons correspondant au chiffre suivant :

```
        JMP   DIGIT
```

La routine DELAY est classique :

```
DELAY   LDA   #DELCON
WAIT    SEC
        SBC   #$01
        BNE   WAIT
        RTS
```

La table d'équivalence, qui fournit les demi-périodes des sons à générer, apparaît à la fin du programme figure 4-41.

On peut calculer les demi-périodes correspondant à chacune des fréquences, sachant que sept fréquences sont à générer : 697, 770, 852, 941, 1 209, 1 336, 1 477.

Par exemple, pour une fréquence N = 697 Hz, la période est de 1/N = 1 434,7 microsecondes ; la demi-période est donc de 717 microsecondes, soit 02CD en hexadécimal.

FRÉQUENCE DÉSIRÉE	DEMI-PÉRIODE	N = DEMI-PÉRIODE-1.7	HEXA pour Ex. 4-4
697	717.3	716	02CC
770	649.3	648	0288
852	586.8	585	0249
941	531.3	530	0212
1209	413.5	412	019C
1336	374.2	372	0174
1477	338.5	337	0151

Fig. 4-44 : Calcul des constantes de temps

De même, pour les autres fréquences, les demi-périodes apparaissent figure 4-44. Les valeurs hexadécimales correspondantes ont déjà été utilisées dans le programme de la figure 4-41.

Examinons maintenant quelques améliorations possibles de ce programme de base.

Exercice 4-4 : *La précision des fréquences produites peut être légèrement augmentée : en se reportant au chapitre 2 du présent ouvrage aussi bien qu'à un manuel hardware, on remarquera que,*

en mode oscillateur, le son généré par T1 n'a pas exactement la fréquence prévue. En fait, il ajoute 1,5 ou 2 µs à la valeur de la demi-période chargée dans le compteur. Vous recalculerez les demi-périodes à utiliser, en supposant que les deux temporisateurs ajoutent en moyenne 1,75 µs à la demi-période.

Note : Ne regardez pas tout de suite, mais la réponse se trouve à la figure 4-44.

Exercice 4-5 : *Ce programme peut, aussi, être amélioré, d'un point de vue fonctionnel, en ajoutant un symbole de "silence" programmable. Ce détail peut être utile dans certains pays dans le cas d'appels internationaux ou bien dans une entreprise, pour accéder à une ligne extérieure. Il faut d'abord composer quelques chiffres pour obtenir une ligne, puis attendre un temps déterminé, avant de composer le numéro proprement dit. Vous incorporerez cette modification dans le programme ci-dessus.*

Nous montrons ci-dessous une amélioration hardware qui permet d'obtenir des fréquences plus propres.

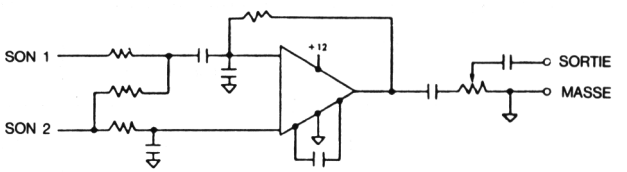


Fig. 4-45 : Suggestion d'amélioration hardware pour obtenir des fréquences plus propres

SECTION 2 : COMBINAISONS DES TECHNIQUES ÉLÉMENTAIRES

INTRODUCTION

Les programmes présentés dans cette section feront appel à une combinaison des techniques exposées au début de ce chapitre. Ils seront développés pour la carte KIM. La seule différence entre le KIM et le SYM, en ce qui concerne ces exercices, portera sur la localisation des PIO dans la carte d'implantation mémoire. Le lecteur intéressé retrouvera à la figure 3-4 la carte mémoire complète du KIM. Dans la mesure où ces programmes sont écrits en langage assembleur, utilisant des étiquettes et des opérandes symboliques, on peut affirmer que la plupart d'entre eux seraient

identiques sur le SYM. Ce n'est que lors du processus d'assemblage (réalisé soit manuellement, soit à l'aide d'un assembleur automatique comme celui présenté dans l'appendice A), c'est-à-dire au moment où est générée la représentation hexadécimale des instructions que vont apparaître les différences dues au caractère hétérogène des implantations-mémoire.

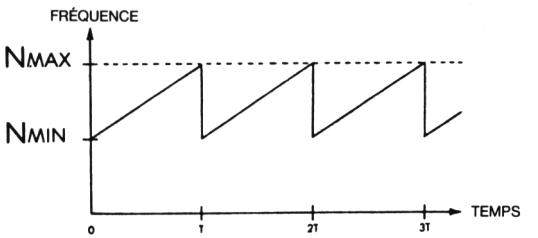


Fig. 4-46 : Son d'une sirène

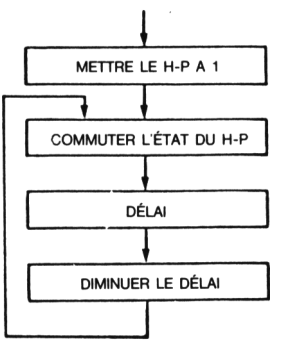


Fig. 4-47 : Ordinogramme pour la sirène ; rampe ascendante

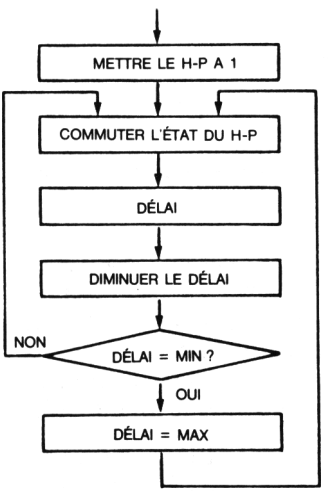


Fig. 4-48 : Arrêt à NMAX

SIRÈNE

La figure 4-46 donne la représentation graphique du son d'une sirène. Le son commence à la fréquence minimum Nmin, et augmente, pendant le temps T, jusqu'à une valeur maximum appelée Nmax. La fréquence du son retombe alors instantanément à Nmin, puis remonte jusqu'à l'instant 2T, et ainsi de suite. L'ordinogramme de production d'un son de fréquence croissante apparaît figure 4-47, en outre la fréquence ne doit pas dépasser Nmax, faute de quoi le son deviendrait inaudible, à moins qu'il ne puisse plus être produit par le haut-parleur. L'ordinogramme qui convient pour générer répétitivement une rampe apparaît figure 4-48.

Le programme apparaît figure 4-49. Il réalise une approximation de la courbe de la figure 4-46.

```
                    ;SIREN
                    ;
                    PA        =$1700
                    PAD       =$1701
                    ;
0000: FF            DELAY     .BYT $FF
                              .=$40
0040: A9 01                   LDA #$01
0042: 8D 01 17                STA PAD
0045: 8D 00 17                STA PA
0048: EE 00 17      SWITCH    INC PA
004B: A6 00                   LDX DELAY
004D: CA            LOOP      DEX
004E: D0 FD                   BNE LOOP
0050: C6 00                   DEC DELAY
0052: 4C 48 00                JMP SWITCH
                    ;

SYMBOL TABLE:
 PA          1700          PAD         1701          DELAY       0000
 SWITCH      0048          LOOP        004D
DONE
```

Fig. 4-49 : Programme de sirène conforme à l'ordinogramme de la fig. 4-47

Le haut-parleur est relié au registre IORA (adresse-mémoire 1 700 hexa), à la position du bit 0. Il peut être connecté directement, mais le circuit de la figure 4-50 vous permettra d'obtenir un meilleur son.

Il est nécessaire de configurer le registre direction DDRA de ce PIO, pour que le BIT 0 soit une sortie :

```
LDA   #$01
STA   PAD          DDRA
```

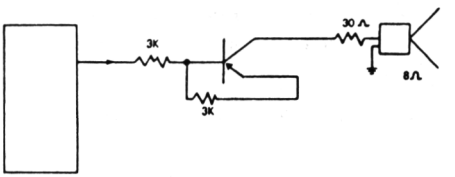


Fig. 4-50 : Branchement amélioré du haut-parleur

On peut alors mettre à 1 le haut-parleur. La mise alternative à un et à zéro du haut-parleur peut se faire facilement, grâce à une astuce de programmation, qui consiste à utiliser l'instruction INC. Cette instruction incrémente le contenu du registre désigné, et génère des nombres successifs, alternativement pairs et impairs. De cette façon, le bit le plus à droite (bit 0 auquel le haut-parleur est relié) passera de la valeur 0 à la valeur 1, et de 1 à 0. L'état du haut-parleur pourrait être commuté en une seule instruction, au lieu de deux, s'il fallait charger un autre motif dans l'accumulateur, avant de le transférer dans ORA. Nous mettrons le haut-parleur à la position 0, où il restera pendant une durée déterminée par la constante "DELAY". La boucle de délai est la suivante :

```
        STA   PA        VALEUR INITIALE DANS
                        ORA
SWITCH  INC   PA        COMMUTE
        LDX   DELAY
LOOP    DEX             BOUCLE DE DÉLAI
        BNE   LOOP
```

Lorsque la valeur DELAY a été utilisée, on la décrémente :

```
        DEC   DELAY
```

De cette façon, au prochain passage la valeur du délai sera plus petite, et le son plus aigu. On commutera à nouveau le haut-parleur :

```
        JMP   SWITCH
```

Le programme ci-dessus réalise la montée du son de la sirène conformément à l'ordinogramme de la figure 4-47.

Exercice 4-7 : *Complétez le programme, en fonction de l'ordinogramme de la figure 4-48, afin de générer des rampes successives, et de produire un son de sirène véritable.*

Exercice 4-8 : *Ecrivez un programme de sirène dont le son monte, descend (graduellement), puis remonte, etc.*

RÉCEPTION D'UNE IMPULSION

Le programme devra mesurer la durée pendant laquelle on appuie sur un bouton, et faire ensuite retentir *n* bip-bip dans le haut-parleur, où *n* sera le temps d'appui exprimé en secondes. Le haut-parleur est connecté au bit 0 de IORA, comme dans le programme précédent. L'interrupteur est connecté au bit 7 de IORA, ce qui facilite la détection. Le branchement est illustré figure 4-51.

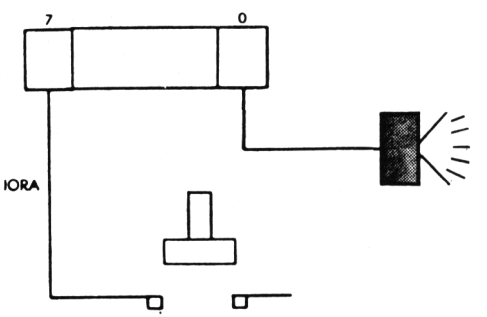


Fig. 4-51 : Branchement de l'interrupteur et du haut-parleur

L'ordinogramme du programme apparaît figure 4-52. La durée pendant laquelle l'interrupteur est fermé est mesurée en quarts de secondes, puis convertie en secondes. On active alors le haut-parleur.

Le programme apparaît figure 4-53. Il suit l'ordinogramme précédent, et devrait aller de soi.

MESURE D'IMPULSION

Nous mesurerons le temps pendant lequel un bouton sera maintenu appuyé, et produirons un son de fréquence proportionnelle à la durée de fermeture de l'interrupteur.

NOTES :
- COMPTEUR contient le nombre « n » de bip-bip
- N est une durée

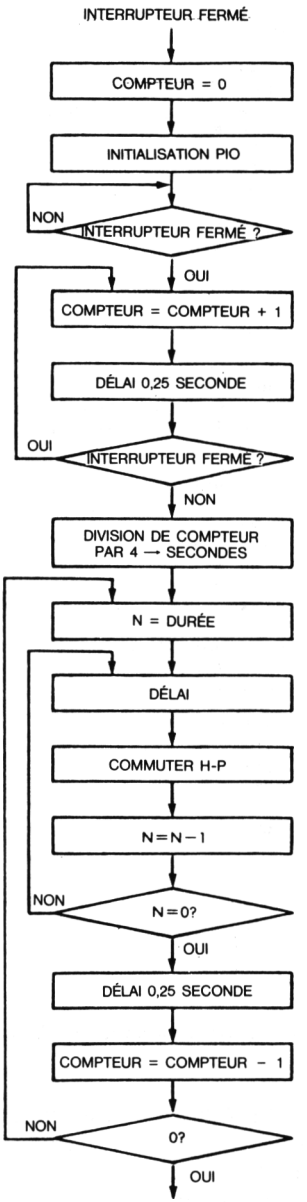


Fig. 4-52 : Ordinogramme détaillé

```
                    T       =$00
                    PA      =$1700
                    PAD     =$1701
                    ;
                            .=$40
0040: A9 01                 LDA #01
0042: 8D 01 17              STA PAD         ;PA0 IS OUTPUT
0045: A9 00                 LDA #0
0047: 85 00                 STA T
0049: 8D 00 17              STA PA
004C: AD 00 17      POL     LDA PA
004F: 30 FB                 BMI POL         ;SWITCH UP?
0051: E6 00         CPT     INC T
0053: A2 3D                 LDX #$3D        ;.25 SEC DELAY
0055: A0 00         BL2     LDY #0
0057: C8            BL1     INY
0058: D0 FD                 BNE BL1
005A: E8                    INX
005B: D0 F8                 BNE BL2
005D: AD 00 17              LDA PA
0060: 10 EF                 BPL CPT
                    ;SWITCH IS UP: RING SPEAKER ONCE.
0062: 46 00                 LSR T           ;DIVIDE BY FOUR
0064: 46 00                 LSR T
0066: A9 00         SOUND   LDA #0
0068: A2 80                 LDX #$80
006A: A0 00         CL2     LDY #00
006C: C8            CL1     INY
006D: D0 FD                 BNE CL1
006F: 49 01                 EOR #1
0071: 8D 00 17              STA PA
0074: E8                    INX
0075: D0 F3                 BNE CL2
                    ;NEW 1/4 SECOND DELAY
0077: A2 3D                 LDX #$3D
0079: A0 00         DL2     LDY #00
007B: C8            DL1     INY
007C: D0 FD                 BNE DL1
007E: E8                    INX
007F: D0 F8                 BNE DL2
0081: C6 00                 DEC T
0083: 10 E1                 BPL SOUND
0085: 00                    BRK

SYMBOL TABLE:
 T          0000        PA          1700        PAD         1701
 POL        004C        CPT         0051        BL2         0055
 BL1        0057        SOUND       0066        CL2         006A
 CL1        006C        DL2         0079        DL1         007B

DONE
```

Fig. 4-53 : Programme de mesure de la durée de fermeture d'un interrupteur

L'ordinogramme de ce programme est, pour l'essentiel, analogue au précédent. Il apparaît figure 4-54. Le programme correspondant se trouve à la figure 4-55.

Nous utiliserons le sous-programme DELAY, qui marque un délai de 0,25 seconde. L'ordinogramme de ce sous-programme apparaît figure 4-56. Le programme correspondant est listé figure 4-57.

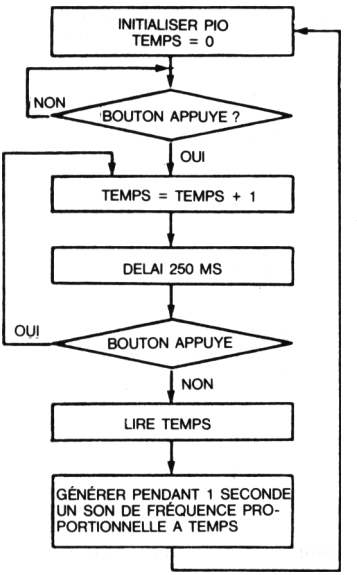


Fig. 4-54 : Mesure du temps de fermeture

```
                    PA      =$1700
                    PAD     =$1701
                    DL250   =$0090
                    FREQ    =$00C0
                    ;
                           .=00
0000: 00            T       .BYT $00
                    ;
                           .=$40
0040: A9 00                 LDA #00
0042: 85 00                 STA T          ;INIT TIME
0044: 8D 00 17              STA PA
0047: A9 01                 LDA #01
0049: 8D 01 17              STA PAD        ;BIT 0 OUT
004C: AD 00 17      POL     LDA PA         ;POLLING...
004F: 30 FB                 BMI POL        ;NOT PRESSED,
0051: E6 00         CPT     INC T          ;INCREMENT TIME
0053: 20 90 00              JSR DL250      ;250 MS DELAY.
0056: AD 00 17              LDA PA
0059: 10 F6                 BPL CPT
005B: A5 00         HERE    LDA T
005D: 0A                    ASL A          ;MPY BY TWO
005E: 0A                    ASL A          ;MPY BY TWO AGAIN
005F: 20 C0 00              JSR FREQ       ;MAKE TONE.
0062: 4C 5B 00              JMP HERE


SYMBOL TABLE:
 PA          1700        PAD         1701        DL250       0090
 FREQ        00C0        T           0000        POL         004C
 CPT         0051        HERE        005B
DONE
```

Fig. 4-55 : Le programme de mesure du temps de fermeture : génération du son

```
                ;MAKES A TONE, USES REG. A
                ;ASSUMES PA SET FOR OUTPUT,
                ;FREQUENCY CONSTANT IN REG. A ON ENTRY
                ;
                PA      =$1700
                F       =$BF
                ;
                        .=$C0
00C0: 85 BF     FREQ    STA F
00C2: A9 00             LDA #0
00C4: A2 80             LDX #$80        ;DURATION CONSTANT
00C6: A4 BF     FL2     LDY F           ;FREQUENCY CONSTANT IN Y
00C8: C8        FL1     INY
00C9: D0 FD             BNE FL1
00CB: 49 01             EOR #1
00CD: 8D 00 17          STA PA          ;TOGGLE PA0
00D0: E8                INX
00D1: D0 F3             BNE FL2
00D3: A5 BF             LDA F
00D5: 60                RTS


SYMBOL TABLE:
 PA             1700            F               00BF            FREQ            00C0
 FL2            00C6            FL1             00C8
DONE
```

Fig. 4-55 : Le programme de mesure du temps de fermeture (suite)

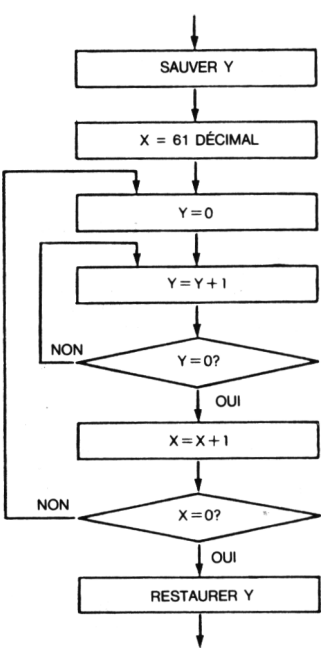


Fig. 4-56 : Ordinogramme du délai 250 ms

```
                ;***** DL250 *****
                ;250 MILLISECOND DELAY
                ;REG. Y UNAFFECTED
                ;
                        .=$90
0090: 98        DL250   TYA             ;SAVE Y
0091: A2 3D             LDX #$3D
0093: A0 00     DL2     LDY #0
0095: C8        DL1     INY             ;INNER LOOP
0096: D0 FD             BNE DL1
0098: E8                INX
0099: D0 F8             BNE DL2         ;OUTER LOOP
009B: A8                TAY             ;RESTORE Y
009C: 60                RTS


SYMBOL TABLE:
 DL250        0090          DL2          0093          D 1          0095

DONE
```

Fig. 4-57 : Délai de 250 ms

Exercice 4-9 : *L'ordinogramme de la figure 4-56 a été écrit de façon que chaque rectangle corresponde à une instruction de la figure 4-57. A l'aide de cet ordinogramme, ou du programme, écrivez, à gauche de chaque rectangle, la durée du délai qu'il introduit. Calculez la durée interne totale du sous-programme qui en résulte. Est-elle exactement de 250 ms ?*

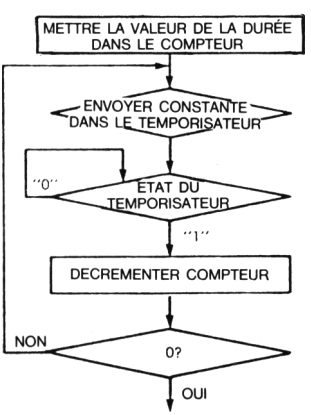


Fig. 4-58 : Ordinogramme de TIME10

```
                    ;***** TIME10 *****
                    ;1/10 SECOND DELAY
                    ;
                    TIMER     =$1707
                    D         =$9D
                    ;
                              .=$9E
009E: 86 9D         TIME10    STX D
00A0: A9 62         T0        LDA #$62     ;DECIMAL 98
00A2: 8D 07 17              STA TIMER
00A5: AD 07 17      T1        LDA TIMER
00A8: 10 FB                 BPL T1
00AA: C6 9D                 DEC D
00AC: D0 F2                 BNE T0
00AE: 60                    RTS

SYMBOL TABLE:
 TIMER          1707          D             009D          TIME10        009E
 T0             00A0          T1            00A5
```

Fig. 4-59 : Délai de 0,1 seconde

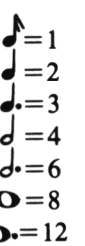
PROGRAMME DE MUSIQUE SIMPLE

Avant de nous lancer dans la musique, nous générerons, à titre préliminaire, un son dans le haut-parleur, à l'aide d'un délai programmé. L'ordinogramme est montré figure 4-58, et le sous-programme de délai figure 4-59. En premier lieu, il est nécessaire de charger la constante F avec la durée appropriée, qui déterminera la fréquence du son.

Pour produire de la musique présentant quelque ressemblance avec des airs réels, il faut émettre un son de fréquence déterminée, et contrôler sa durée. Les notations musicales utilisées pour indiquer la durée d'un son sont énumérées ci-après :

Notation musicale

(. = + 50 %)



Le point qui peut éventuellement suivre une note, augmente sa durée de 50 %. Il y a schématiquement sept durées possibles. En outre, il faut se donner les moyens de représenter un « silence ». Au minimum cette information nécessite 3 bits en format codé, ou 4 bits en format *décodé*. (Dans un format décodé, les valeurs 1, 2, 3, 4, 6, 8 et 12 sont représentées par leur valeur proprement dite, en binaire.)

Pour représenter une octave, il faut prendre en compte les 7 notes (do, ré, mi, fa, sol, la, si), ainsi que les 6 demi-tons intermédiaires. Soit, au total, 13 touches. Pour représenter plus d'une octave, il sera nécessaire d'allouer tout un octet pour représenter la hauteur de la note. Si le lecteur ne dispose, sur sa carte, que d'une mémoire limitée, il souhaitera peut-être se restreindre à 16 touches ; il aurait, dans ce cas, la possibilité d'utiliser un code, où la moitié gauche de chaque octet représente la durée, et la partie droite, la hauteur de la note.

Notre ambition se limite à jouer des airs très peu élaborés, faisant appel à un codage simple, où un octet sera alloué à la durée, et un deuxième octet à la fréquence de la note. Les figures 4-60 à 4-62 présentent trois exemples : *Sonatine* de Mozart, *Choral* de Bach et *Au clair de la Lune.*

L'ordinogramme du programme de musique correspondant est montré figure 4-63, et le programme lui-même figure 4-64. TIME10 est une simple routine préliminaire, qui marque un délai de 0,1 s (figure 4-58 et 4-59).

Adresse	Durée	F	Note
00	09	20	*la*
2	04	4F	*do#*
4	04	6B	*mi*
6	05	12	sol#
8	01	20	*la*
A	01	39	*si*
C	0F	20	*la*
E	02	00	
12	09	7C	*fa#*
12	04	6B	*mi*
14	04	91	*la*
16	04	6B	*mi*
18	04	59	*ré*
1A	09	4F	*do#*
1C	00	00	
1E			
20			

Fig. 4-60 : Sonatine de Mozart

Adresse	Durée	F	Note
00	88	44	*do*
02	06	59	*ré*
04	06	6B	*mi*
06	88	83	*sol*
08	06	74	*fa*
A	06	74	*fa*
C	88	91	*la*
E	06	83	*sol*
10	06	83	*sol*
12	88	A3	*do*
14	06	9E	*si*
16	06	A3	*do*
18	06	83	*sol*
1A	06	6B	*mi*
1C	06	44	*do*
1E	88	59	*ré*
20	06	6B	*mi*
22	06	20	*la*
24	88	83	*sol*
26	06	74	*fa*
28	06	6B	*mi*
2A	88	59	*ré*
2C	06	44	*do*
2E	06	04	*sol*
30	06	44	*do*
32	88	39	*si*
34	06	44	*do*
36	06	6B	*mi*
38	06	83	*sol*
3A	0E	A3	*do*
3C	0E	44	*do*
3E	00	00	

Fig. 4-61 : Choral de Bach

Adresse	Durée	F	Note
0	04	44	*do*
2	04	44	*do*
4	04	44	*do*
6	04	59	*ré*
8	09	6B	*mi*
A	09	59	*ré*
C	04	44	*do*
E	04	6B	*mi*
10	04	59	*ré*
12	04	59	*ré*
14	09	44	*do*
16	10	00	
18	04	44	*do*
1A	04	44	*do*
1C	04	44	*do*
1E	04	59	*ré*
20	09	6B	*mi*
22	09	59	*ré*
24	04	44	*do*
26	04	6B	*mi*
28	04	59	*ré*
2A	04	59	*ré*
2C	09	44	*do*
2E	00	00	

Fig. 4-62 : « Au clair de la lune »

Exercice 4-10 : *Vérifier si la routine réalise exactement un délai de 0,1 seconde. Vérifier la durée de chaque instruction, et précisez combien de fois la boucle sera exécutée. Calculez le délai correspondant.*

CONTRÔLE DE CARREFOUR AVEC UN KIM

La figure 4-66 montre un montage susceptible de simuler un contrôle de circulation routière. Il est équipé, dans chaque voie, d'interrupteurs qui serviront à indiquer la présence d'une voiture, ou qu'un piéton demande le passage.

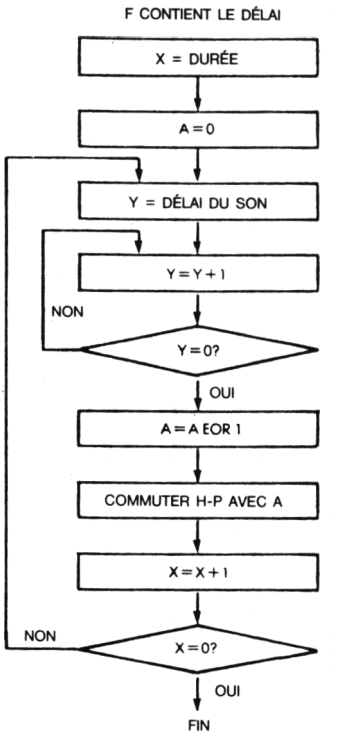


Fig. 4-63 : Ordinogramme pour jouer un air

```
                        ;***** PLAY A TUNE *****
                        PA      =$1700
                        PAD     =$1701
                        TIMER   =$1707
                                .=00
                        ADDRS   .=.+2
                        TEMP    .=.+1
                        YSAVE   .=.+1
                        F       .=.+1
                        ;
                                .=$10
0010: A9 31             TIME20  LDA #$31
0012: 8D 07 17                  STA TIMER
0015: 2C 07 17          T1      BIT TIMER
0018: 10 FB                     BPL T1
001A: CA                        DEX
001B: D0 F3                     BNE TIME20
001D: 60                        RTS
```

Fig. 4-64 : Programme qui joue un air

```
                              .=$20
0020: 84 03      FREQT        STY YSAVE
0022: 85 04                   STA F
0024: A9 31      FT0          LDA #$31
0026: 8D 07 17                STA TIMER      ;START TIMER (1/20 SEC.)
0029: A4 04      FT1          LDY F
002B: C8         FT2          INY
002C: D0 FD                   BNE FT2
002E: EE 00 17                INC PA         ;SWITCH SPEAKER
0031: 2C 07 17                BIT TIMER      ;TIME ELAPSED?
0034: 10 F3                   BPL FT1        ;NO: GO ON.
0036: CA         FT3          DEX
0037: D0 EB                   BNE FT0
0039: A4 03                   LDY YSAVE
003B: 60                      RTS

                              .=$40
0040: A2 0F      START        LDX #$0F
0042: 9A                      TXS
0043: A9 00                   LDA #$00
0045: 8D FA 17                STA $17FA
0048: 8D FE 17                STA $17FE
004B: A9 1C                   LDA #$1C
004D: 8D FB 17                STA $17FB
0050: 8D FF 17                STA $17FF      ;INTERRUPT VECTOR
0053: A9 01                   LDA #$01
0055: 8D 01 17                STA PAD        ;PAD IS OUTPUT
0058: A0 00      DACAPO       LDY #$00
005A: B1 00      NEXT         LDA (ADDRS),Y
005C: 85 02                   STA TEMP
005E: 29 7F                   AND #$7F
0060: AA                      TAX            ;DURATION
0061: F0 F5                   BEQ DACAPO
0063: C8                      INY
0064: B1 00                   LDA (ADDRS),Y
0066: F0 10                   BEQ TONE
0068: 20 29 00                JSR FREQT
006B: 24 02                   BIT TEMP
006D: 30 05                   BMI AFTER
006F: A2 02                   LDX #$02
0071: 20 10 00                JSR TIME20
0074: C8         AFTER        INY
0075: 4C 5A 00                JMP NEXT
0078: 20 10 00   TONE         JSR TIME20
007B: F0 F7                   BEQ AFTER

SYMBOL TABLE:
 PA          1700      PAD         1701      TIMER       1707
 ADDRS       0000      TEMP        0002      YSAVE       0003
 F           0004      TIME20      0010      T1          0015
 FREQT       0020      FT0         0024      FT1         0029
 FT2         002B      FT3         0036      START       0040
 DACAPO      0058      NEXT        005A      AFTER       0074
 TONE        0078
```

Fig. 4-64 : Programme qui joue un air (suite)

Exercice 4-11 : *Ecrivez un programme de contrôle de carrefour qui suive les spécifications suivantes :*

- *Durée minimum du feu orange : 3 secondes.*
- *Chaque fois qu'on détecte la présence d'une voiture (en maintenant pressé un des boutons) faire durer le feu vert, pour la rue concernée, pendant 3 secondes + durée de la présence.*
- *Durée maximum du feu vert pour n'importe quelle direction : 2 minutes, s'il y a une demande de la direction opposée.*
- *Régime « orange clignotant la nuit » (la nuit est indiquée par un interrupteur supplémentaire).*
- *Un ordinogramme possible pour ce programme apparaît figure 4-65. Ecrivez le programme correspondant.*

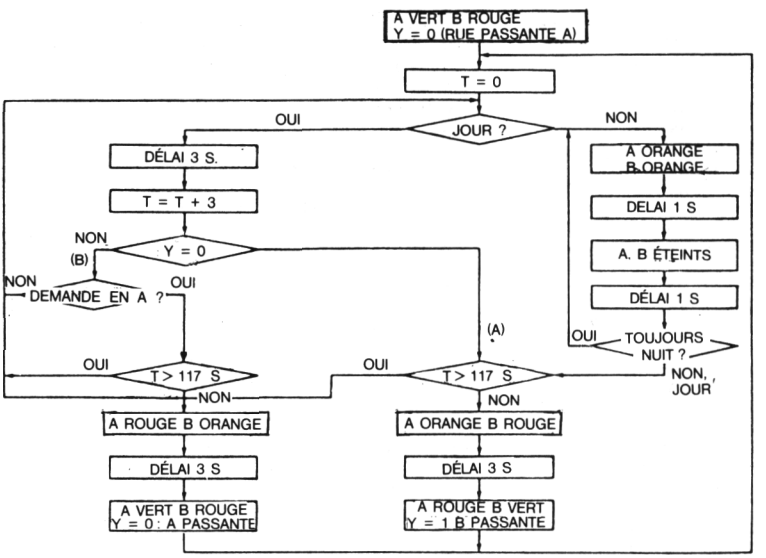


Fig. 4-65 : Ordinogramme contrôle de carrefour

JEU SUR LA TABLE DE MULTIPLICATION

Comme exercice final, nous proposerons un programme d'enseignement de la table de multiplication. Ce programme devra successivement faire clignoter *n* fois, une LED (ou sonner le haut-parleur), *n* étant compris entre 1 et 10, attendre 2 secondes, et faire, à nouveau, clignoter *p* fois, *p* étant compris entre 1 et 10.

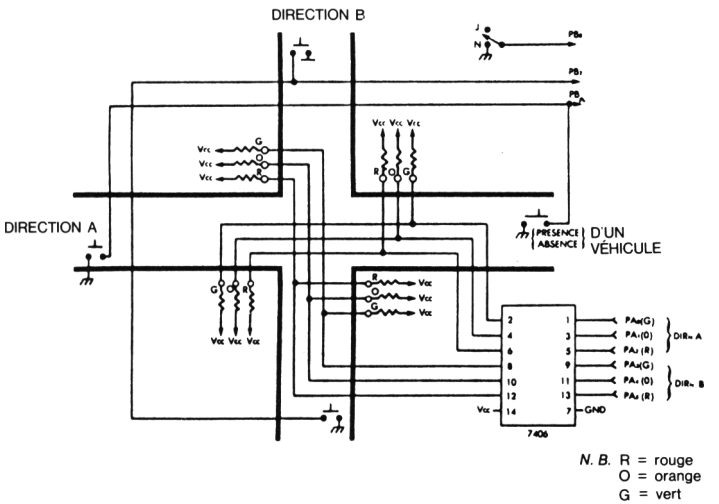


Fig. 4-66 : Contrôleur de carrefour

L'utilisateur devra appuyer $n \times p$ fois sur un bouton pour fournir sa réponse, que le haut-parleur devra acquitter de façon audible. L'utilisateur achèvera sa réponse en s'abstenant de toucher au bouton pendant au moins 3 secondes. Si la réponse est correcte, la LED doit s'allumer pendant 5 secondes. Si elle est fausse, elle doit clignoter.

Exercice 4-12 : *Dessinez l'ordinogramme correspondant, et écrivez le programme. (Le programme est simple, mais un peu plus long que la plupart des précédents. Si vous avez vraiment besoin de la réponse, vous la trouverez dans l'appendice B.)*

RÉCAPITULATION

Nous avons dans ce chapitre, connecté, à une carte 6502, des dispositifs d'entrée-sortie simples. Nous avons appris à réaliser des interfaces hardware simples, et à développer du software d'application simple, pour percevoir et contrôler un environnement. Les applications présentées ici restent toujours très simples. On pourrait, néanmoins, développer des applications plus complexes, basées sur le même hardware. Nous sommes mintenant prêts à aborder les interfaces, et les programmes plus élaborés du chapitre V : Applications industrielles et domestiques.

5

APPLICATIONS INDUSTRIELLES ET DOMESTIQUES

INTRODUCTION

Nous avons présenté, au chapitre IV, les techniques fondamentales permettant de connecter des dispositifs simples à un micro-ordinateur, basé sur le 6502, et de développer le software des applications de base. Nous allons maintenant interfacer, à la carte 6502, des dispositifs plus complexes, et développer des programmes d'applications eux-mêmes sensiblement plus élaborés. Ces applications sont caractéristiques des situations de contrôle industriel ou domestique. Au chapitre suivant, nous interfacerons des périphériques de micro-ordinateur à la carte 6502.

Nous envisagerons d'abord une simulation de contrôle de carrefour. Les feux de circulation seront simulés sur la carte par des LED. Les programmes développés seront de complexité croissante, allant jusqu'à permettre de détecter la présence de voitures en attente, en simulant, par des boutons-poussoirs, les détecteurs à boucle d'induction normalement encastrés dans la chaussée. Les connaissances requises pour développer ces interfaces hardware et software ne sont pas différentes de celles mises en œuvre dans un environnement de contrôle industriel réel.

Ensuite, nous interfacerons à ce système, une LED à matrice de points 5×7. Cette technique est fréquemment utilisée pour afficher des données. Les matrices de points peuvent également servir à représenter des caractères sur un écran de télévision, ou sur une imprimante à matrice. Nous aurons recours à cette application pour afficher les valeurs d'interrupteurs connectés à la carte 6502.

L'étape suivante consistera à générer des sons avec le haut-parleur, afin de développer des programmes de musique simples.

Nous nous servirons de l'ensemble d'interrupteurs pour spécifier la note qui doit être jouée. Les connaissances acquises en contrôlant le son du haut-parleur seront également utilisées par le programme suivant, qui produit des sons tels que celui d'une sirène.

Nous nous intéresserons, par la suite, à un système d'alarme antivol, à l'usage d'une maison ou d'un immeuble. L'un des dispositifs de détection d'une éventuelle intrusion sera constitué par un faisceau lumineux, dont l'interruption provoquera une alarme par l'intermédiaire du haut-parleur. A partir de ce point de départ, de nombreuses améliorations seront proposées en exercice.

Autre application : on régulera par ordinateur la vitesse d'un moteur ordinaire à courant continu, opération, en fait, très simple, avec l'aide de techniques digitales. Nous présenterons ces techniques, ainsi que l'interface hardware nécessaire.

Nous connecterons, par ailleurs, un capteur de température au micro-ordinateur. La mesure de température sera traduite sous forme d'un son audible. Plus haute sera la température, plus aigu sera le son. D'où l'introduction du concept de conversion analogique-numérique. Les techniques hardware et software réelles, utilisées pour effectuer cette conversion, seront présentées.

Le lecteur est engagé à construire la carte d'applications n° 2 utilisée par les programmes de ce chapitre. Tous les composants utilisés sur cette carte sont de prix modique et, en principe, immédiatement disponibles chez les revendeurs d'électronique (sauf, peut-être, le convertisseur digital-analogique, qui doit souvent être commandé chez un distributeur). Les figures 5-1, 5-2 et 5-3 montrent des photographies de cette carte.

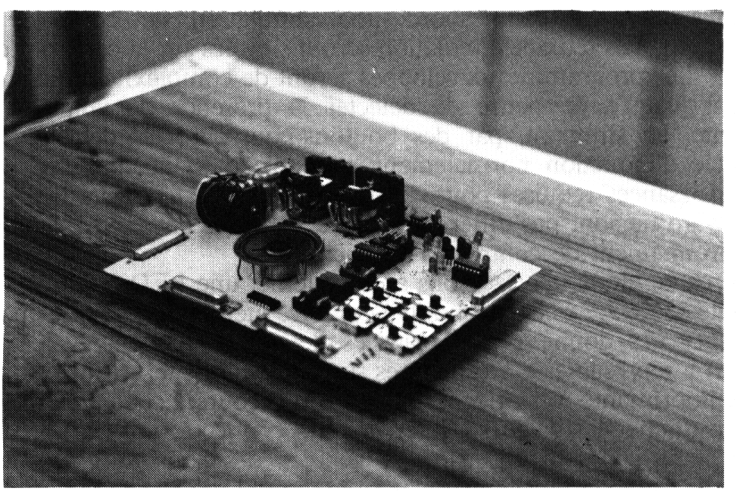


Fig. 5-1 : La carte d'application n° 2

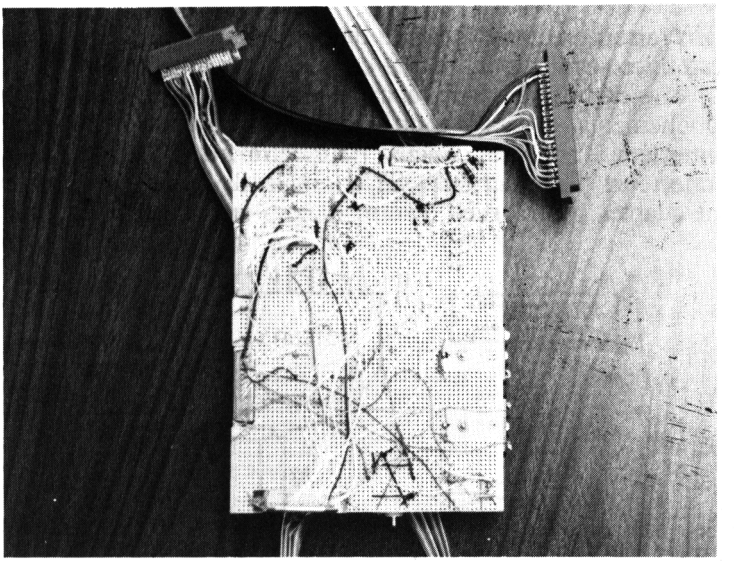


Fig. 5-2 : Sur le dessous, on voit le wrapping (connexions enroulées)

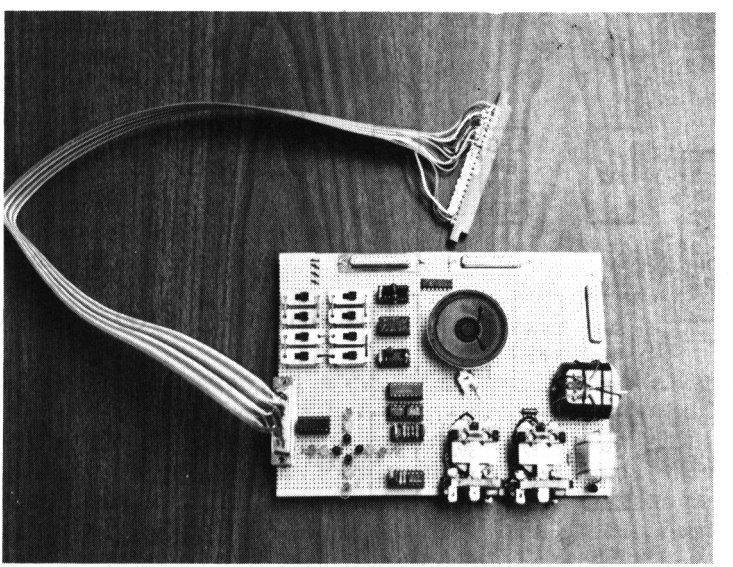


Fig. 5-3 : La carte est connectée par des câbles plats

En raison du nombre limité de ports disponibles, en sortie de la carte micro-ordinateur, quatre connecteurs, appelés H1, H2, H3 et H4, ont été installés sur la carte d'applications pour faciliter les branchements, et éviter tout recâblage entre les programmes. Ces connecteurs ont été conçus pour être reliés directement aux connecteurs extérieurs du SYM, mais ils pourraient facilement être adaptés aux sorties d'autres micro-ordinateurs. Pour chaque

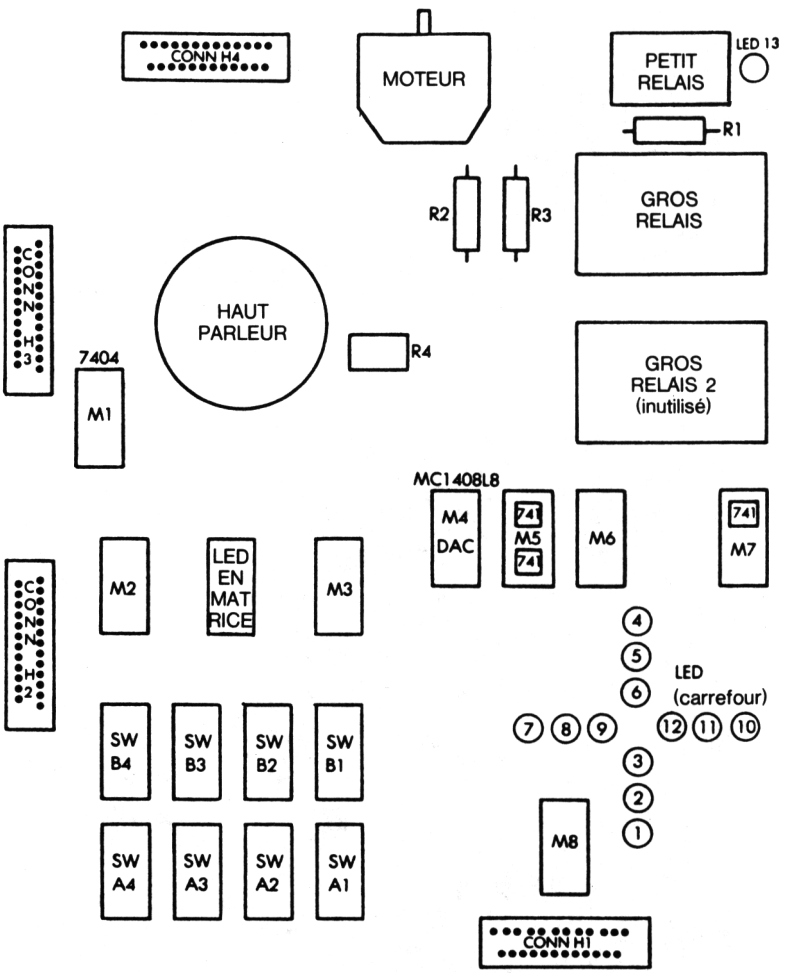


Fig. 5-4 : Schéma de la carte d'applications

application, il sera nécessaire de brancher un ou deux câbles, venant du micro-ordinateur, sur les connecteurs correspondants de la carte d'applications. Les détails de chaque connexion seront fournis au début de chaque application.

L'emplacement des composants sur la carte est donné par la figure 5-4, et les détails des connecteurs par les figures 5-5 a et b. Les détails de chaque branchement seront donnés dans le paragraphe correspondant à chaque application.

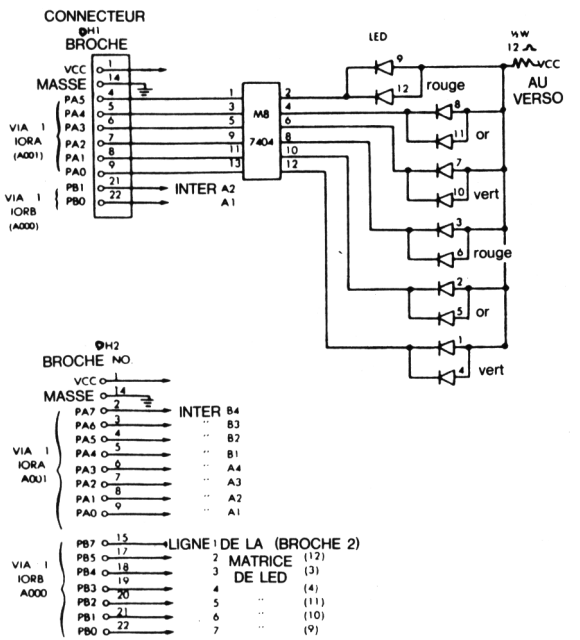


Fig. 5-5a : Connecteurs H1 et H2

La technique du wrapping a été utilisee pour câbler la carte, comme le montre la figure 5-2. Il est naturellement possible de souder les fils. N'oubliez pas les précautions habituelles de manipulation des circuits LSI : tous les instruments (et vous-même) doivent être reliés à la terre. Dernier détail, le potentiomètre (résistance variable), branché en série avec le haut-parleur, ne doit pas être mis à zéro. S'il l'était, il risquerait d'être détruit, dans l'hypothèse où on connecterait le haut-parleur à une des sorties amplifiées (ce qui est le cas du SYM). De plus, le transistor de sortie serait probablement détruit lui-aussi. Il est donc recommandé de

placer une résistance supplémentaire en série avec le haut-parleur.

Le but de ce chapitre est d'enseigner les techniques d'applications réelles et par là même, de vous rendre capables, soit de créer des applications domestiques de complexité non négligeable, soit de résoudre de véritables problèmes de contrôle industriel dans un environnement réel. Sa lecture devrait, en principe, vous permettre d'acquérir toutes les connaissances fondamentales nécessaires pour commencer à traiter de vous-même des applications complexes. Pour les problèmes spécifiques d'interfaçage nous vous renvoyons à la référence C5 : *Techniques d'interface aux microprocesseurs.*

Note importante : Pour disposer d'une ligne d'entrée-sortie supplémentaire, on a court-circuité le transistor 1 (le plus près du centre) des quatre ports amplifiés du SYM.

Les programmes présentés dans cette section ont été conçus de telle sorte qu'ils puissent être améliorés. Le lecteur avancé ne manquera pas de remarquer un grand nombre d'améliorations de style possibles, dont certaines seront proposées ou décrites dans la section « exercices », qui suit chaque application. Lorsque vous lirez les programmes, nous vous conseillons d'être à l'affût de telles améliorations. Nous attendrons, toutefois, le chapitre suivant pour vous présenter des programmes optimisés : une fois que tous les problèmes auront été résolus.

Encore une fois votre intérêt est de vous efforcer de résoudre la plupart des exercices que nous vous soumettons : soit sur papier, soit sur un micro-ordinateur proprement dit. Ils ont été préparés avec soin, de manière à vous permettre de vous assurer que les concepts présentés dans la section précédente ont été réellement compris, et à exciter votre créativité. Si vous résolvez les exercices sans consulter le livre, c'est incontestablement que vous avez appris à résoudre vos propres problèmes d'application.

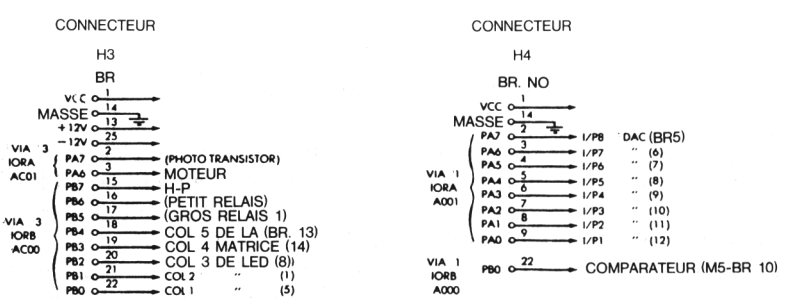


Fig. 5-5b : Connecteurs H3 et H4

SYSTÈME DE CONTRÔLE DE CARREFOUR

Nous allons développer des programmes capables de contrôler un carrefour simulé. La figure 5-6 donne le schéma du carrefour qui comporte deux directions de circulation, appelées A et B. En jargon de spécialiste, cela s'appelle des « phases ». Les deux feux s'adressant respectivement aux deux directions d'une phase (par exemple les deux feux de l'avenue A) sont de la même couleur (vert, orange, rouge), à un instant donné. De même, les feux de la phase B fonctionnent simultanément. Ces quatre ensembles de feux de carrefour seront simulés, sur notre carte, par quatre ensembles de LED vertes, orange et rouges. De plus nous supposerons que des boucles détectrices de véhicules ont été encastrées dans la chaussée, aux endroits marqués A-1, A-2, B-1, B-2, sur le schéma de la figure 5-6.

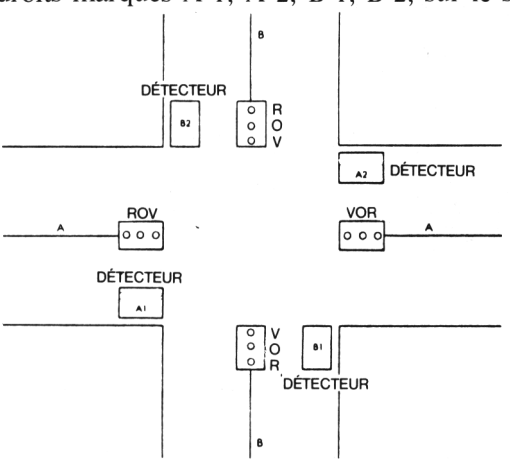


Fig. 5-6 : Le système de contrôle de carrefour

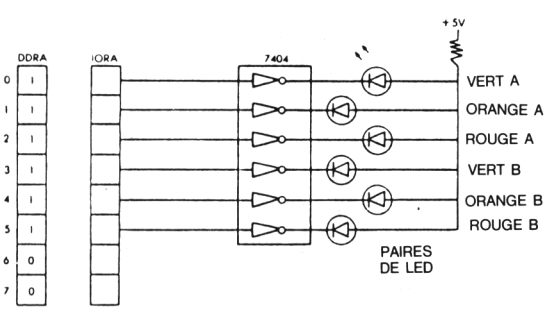


Fig. 5-7 : Connexion des LED

Elles seront appelées « boucles détectrices ». Leur rôle sera expliqué plus tard.

Examinons le branchement hardware de nos « feux de carrefour » (en fait, les LED) munis d'un système à microprocesseur. D'après la figure 5-7, on connecte un amplificateur 7404 au registre IORA du 6522 n° 1. Les paires de LED apparaissent à droite de l'illustration. Pour plus de clarté, nous n'avons représenté qu'une LED sur chaque ligne. En fait, deux LED sont connectées en parallèle sur chaque ligne, puisqu'il existe deux ensembles de feux pour chaque phase. Le branchement effectif est montré figure 5-8. Pour configurer en sorties les 6 bits les plus bas de IORA on a chargé dans le registre direction, DDRA, qui apparaît à gauche de la figure, le motif binaire approprié : "00111111". L'amplificateur (7404) est nécessaire pour fournir une intensité de courant suffisante pour allumer les LED.

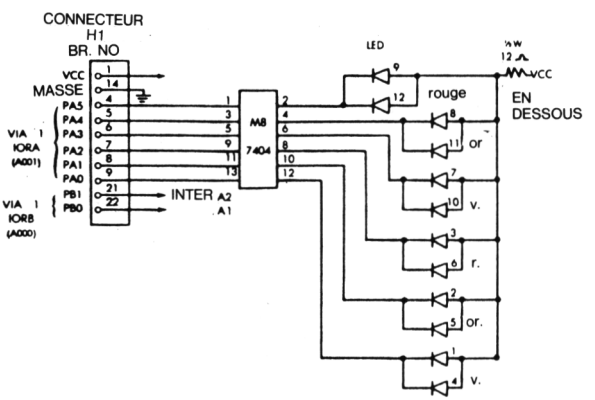


Fig. 5-8 : Branchement effectif des LED

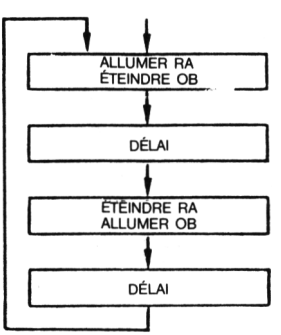


Fig. 5-9 : Le cas de la nuit

Nous développerons maintenant des programmes pour différents algorithmes de contrôle du carrefour. On peut distinguer deux cas principaux : le cas de la nuit (feux clignotants) et le cas du jour.

(Branchement : connecteur A sur H1)

```
0100  A9 3F      NIGHT  LDA  #$3F     Met 3F dans DDRA du VIA n° 1 pour le
0102  8D 03 A0          STA  $A003       mettre en sortie

0105  A9 02      NIT2   LDA  #$02
0107  8D 01 A0          STA  $A001     Allume l'orange dans une direction
010A  A9 FC              LDA  $FC       Met le compte pour DLYA à $FC :
010C  85 00              STA  $00         c. à d. – 4) à l'adresse 0000

010E  20 20 01           JSR  DLYA      Appelle DLYA
0111  A9 20              LDA  #$20      Allume l'orange dans l'autre direction
0113  8D 01 A0           STA  $A001

0116  A9 FC              LDA  #$FC      Met le compte pour DLYA A $FC (à
0118  85 00              STA  $00         l'adresse 0000)

011A  20 20 01           JSR  DLYA      Appelle DLYA
011D  4C 05 01           JMP  NIT2      Recommence
```

Sous-programme DLYA : Ce sous-programme prend le nombre à l'adresse 0000 et boucle jusqu'à ce qu'il soit incrémenté depuis une valeur négative prédéterminée jusqu'à zéro. La valeur prédéterminée sert à contrôler la durée du délai.

```
0120  A2 9D      DLYA   LDX  #$9D
0122  A0 71      LPXA   LDY  #$71
0124  C8         LPYA   INY          ┐ Boucle
0125  C0 00              CPY  #$00    │ de délai
0127  30 FB              BMI  LPYA    ┘ interne
0129  E8                 INX
012A  E0 00              CPX  #$00
012C  30 F4              BMI  LPXA
012E  E6 00              INC  $00     Incrémente le compte à chaque fois
                                        qu'une boucle externe est terminée
0130  A5 00              LDA  $00
0132  C9 00              CMP  #$00
0134  30 EA              BMI  DLYA    Boucle jusqu'à ce que le compte soit 0
0136  60                 RTS
```

Boucle de délai externe

Fig. 5-10 : Simulation de feux de carrefour ; mode nuit (programme 5-1)

Cas de la nuit

C'est le cas le plus simple. Les feux sont clignotants : rouge dans une direction, orange dans l'autre. Cette stratégie est utilisée pour des intersections isolées, avec faible circulation nocturne. Elle correspond, notons-le, à la législation américaine. En France, on aurait plutôt l'orange clignotant pour les deux directions, mais cela ne change presque rien au programme. L'ordinogramme correspondant à cet algorithme apparaît figure 5-9. Il implique que le rouge d'une direction, et l'orange de l'autre, soient allumés ou éteints en opposition de phase. Tous deux sont maintenus allumés ou éteints pendant une période fixe, appelée « DELAY ». Le programme correspondant à cet ordinogramme apparaît figure 5-10. Il comporte un programme principal, appelé NIGHT, et un sous-programme de délai, appelé DLYA. Examinons-le.

En se reportant à la figure 5-7, on constate qu'il faut, en premier lieu, configurer le registre direction du 6522 n° 1, de sorte que les 6 bits les plus bas de IORA soient en sortie, pour commander les LED. Ce registre direction est à l'adresse-mémoire A003, et IORA à l'adresse A001 (cf. fig. 3-6 pour la carte d'implantation mémoire du 6522).

Les deux premières instructions écrivent la valeur souhaitée dans le registre direction :

```
NIGHT    LDA   #$3F
         STA   $A003        ÉCRIT DANS DDRA
```

Il reste à déposer, dans IORA, le motif approprié pour allumer ou éteindre les LED concernées. Le motif binaire de chaque LED est donné par la figure 5-11.

BINAIRE	HEXA	FEU
00000001	01	vert A
00000010	02	orange A
00000100	04	rouge A
00001000	08	vert B
00010000	10	orange B
00100000	20	rouge B

Fig. 5-11 : Motif binaire correspondant à chaque paire de LED

Les deux lignes suivantes du programme allument l'orange, pour A, en déposant la valeur hexadécimale "02" dans le registre IORA :

```
NIT2    LDA   #$02
        STA   $A001        ÉCRIT DANS IORA
```

On doit ensuite respecter un délai, dont la valeur est déposée dans l'accumulateur, puis rangée dans la case mémoire 00, où la routine de délai la trouvera. Le sous-programme DLYA est alors appelé :

```
        LDA   #$FC
        STA   $00
        JSR   DLYA
```

Quand le délai spécifié est écoulé, la valeur hexadécimale "20" est envoyée dans IORA, ce qui a pour effet d'éteindre l'orange de la direction A, et simultanément d'allumer le rouge de la direction B. Comme précédemment, on dépose la valeur du délai à l'adresse "0", et on appelle le sous-programme DLYA :

```
        LDA   #$20
        STA   $A001
        LDA   #$FC
        STA   $00
        JSR   DLYA
```

Après expiration du délai, le programme reboucle à NIT2, où on allume, OA et éteint RB :

```
        JMP   NIT2
```

A ce niveau, le fonctionnement du programme devrait être totalement évident. Examinons le sous-programme. Le principe des boucles de délai est de charger un registre (ou une case-mémoire) avec une valeur, puis de l'incrémenter, ou le décrémenter, jusqu'à une valeur prédéterminée. Le lecteur est probablement familier de la technique de décrémentation (cf. réf. C3), que, pour changer, nous allons employer ici, mais qui nécessite, cependant, quelques instructions supplémentaires. Nous suggérerons une amélioration, à titre d'exercice, à la fin de la section. Le sous-programme de délai apparaît figure 5-12. Le délai à réaliser étant de l'ordre de quelques

dizaines de secondes, il ne pourra être obtenu à l'aide d'une seule boucle. Cette solution nous contraindrait à charger un registre avec la valeur 255 (FF en hexadécimal), et à incrémenter (ou décrémenter) à partir de là. Le délai qui en résulterait ne serait pas suffisant. Pour l'augmenter nous utiliserons des boucles imbriquées : une boucle interne et au moins une boucle externe. Cette dernière sera exécutée chaque fois que la boucle interne aura été épuisée. Examinons le programme. Nous utiliserons le registre X comme compteur pour la boucle externe, en le chargeant avec la valeur initiale 9D qui sera justifiée plus loin :

```
DLYA    LDX   #$9D
```

La seconde instruction chargera le registre Y qui est le compteur de la boucle interne avec la valeur hexadécimale 71.

```
LPXA    LDY   #$71
```

Les trois instructions suivantes forment la boucle interne :

```
LPYA    INY
        CPY   #$00
        BMI   LPYA
```

Y est incrémenté, jusqu'à ce qu'il atteigne la valeur 0. Chaque fois que la boucle interne est épuisée, autrement dit quand Y atteint la valeur 0, le compteur externe X est incrémenté. C'est la 6e instruction du programme :

```
        INX
```

Chaque fois que X est incrémenté, il est comparé à la valeur 0, et tant que celle-ci n'est pas atteinte, on revient au début de la boucle externe LPXA :

```
        CPX   #$00
        BMI   LPXA
```

Le délai qui en résulte est, jusqu'à présent, la durée du délai interne, multipliée par le nombre d'exécutions de la boucle externe.

Chaque fois que cette boucle externe est épuisée, notre compteur général (case mémoire 00) est incrémenté :

```
INC    $00
```

Cela forme une troisième boucle. Le contenu de la case mémoire 00 est testé par rapport à 0, chaque fois qu'il est incrémenté. Quand la valeur 0 est atteinte, on sort du sous-programme. Tant qu'elle ne l'est pas, on retourne à DLYA, c'est-à-dire au début de la boucle précédente. Le processus précédent recommence alors :

```
LDA    $00
CMP    #$00
BMI    DLYA
RTS
```

La figure 5-12 montre la structure générale du programme, avec ses trois boucles imbriquées, et la durée de chaque instruction. Le délai total sera égal au contenu de la mémoire 00, multiplié par le nombre d'exécutions de la boucle extérieure, et multiplié encore par le délai interne. Calculons cette durée totale. Les durées des instructions apparaissent figure 5-12. Examinons d'abord la boucle interne. A chaque exécution on réalise trois instructions, d'une durée totale de 7 microsecondes. Pour simplifier, nous demanderons à cette boucle interne de créer un délai d'environ 1 milliseconde. La boucle externe n° 1 aura la responsabilité de créer un délai de 100 millisecondes (0,1 s).

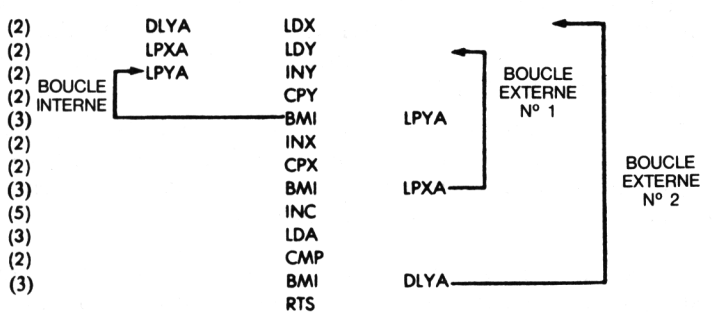


Fig. 5-12 : Organisation des boucles

Commençons avec la valeur "80" (hexadécimal), dans le registre Y. Le milieu de l'étendue qui peut être obtenue avec 8 bits est 128, en décimal. Le parcours de la boucle interne amènera à incrémenter Y 128 fois. La durée de la boucle sera donc $7 \times 128 = 896$ microsecondes. Comme nous voulons obtenir un délai d'environ 1 000 microsecondes, cette valeur N doit être modifiée. Nous la calculerons en sachant qu'elle doit être telle que $N \times 7 = 1\,000$. N doit donc être égal à $1\,000/7 = 142{,}86$. L'entier le plus proche est 143. Dans ce cas particulier, nous incrémenterons Y plutôt que de le décrémenter. Nous chargerons donc, dans Y, la valeur $256 - 143 = 113$ décimal (71 en hexadécimal).

Calculons maintenant le délai introduit par la boucle externe n° 1. Une itération de cette boucle a une durée égale à la durée de la première instruction du programme (en DLYA), à laquelle s'ajoutent, d'une part, la durée de la boucle interne, et d'autre part les trois instructions suivantes, jusqu'à BMI LPXA. Cette durée est de :

$$2 + 7 \times 143 + 7 = 1\,010 \text{ microsecondes.}$$

Nous voulons que cette boucle externe n° 1 crée un délai de 0,1 s ou 100 000 μs. Le nombre P d'exécutions de cette boucle devra être tel que $1\,010 \times P = 100\,000$. P doit donc être égal à $100\,000 : 1010 = 99$. Là encore, nous procéderons par incrémentation, et le nombre à déposer dans X devra être tel que X soit incrémenté 99 fois, avant de retomber sur "00". Le nombre à écrire en X sera égal à $256 - 99 = 157$ en décimal (9D en hexadécimal). Vérifions maintenant la durée totale du délai créé. Le délai de la boucle externe est égal à : $99 \times 1010 = 99\,990$ microsecondes. Les quatre instructions qui restent à exécuter, à la fin du sous-programme DLYA, représentent une durée de $5 + 3 + 2 + 3 = 13$ μs. Il faut encore ajouter 2 μs pour la première instruction de DLYA.

Le délai total pour l'exécution complète de DLYA sera donc : $99\,990 + 15 = 100\,005$ microsecondes. Soit très près de 0,1 s. Si près, qu'on pourrait chronométrer cette routine avec un chronomètre, et vérifier l'exactitude de la méthode.

Une précaution utile : Rappelez-vous que ce sous-programme procède par *incrémentation.* Le nombre déposé à l'adresse 0 contrôlera le nombre de dixièmes de secondes de délai que le sous-programme va introduire. Cependant, le nombre qui sera déposé à l'adresse 00 devra être le complément du nombre de dixièmes de

secondes voulu, car il sera incrémenté, jusqu'à ce qu'il déborde sur 0. En d'autres termes, pour obtenir un délai de 0,4 s, ce n'est pas 4 qu'il faut déposer à l'adresse 0, mais 256 − 4 = 252 décimal = FC hexadécimal. C'est précisément ce que nous avons fait dans le programme de la figure 5-10 (mode nuit).

Le temps est maintenant venu d'améliorer cette routine de délai :

Exercice 5-1 : *Réécrire le sous-programme de délai en procédant par décrémentation, plutôt que par incrémentation. Recalculez les nombres à charger en X et en Y, de sorte que le délai introduit reste approximativement 0,1 seconde. Quel est l'avantage de la décrémentation sur l'incrémentation ?*

Attention : Si vous décidez de procéder par décrémentation n'oubliez pas de changer la valeur FC déposée à l'adresse 00 de la mémoire. Il faut charger une constante différente, avant d'appeler le sous-programme.

Exercice 5-2 : *Modifiez le programme de façon que les feux clignotent à chaque seconde. En outre, raccourcissez-le, en utilisant EOR pour faire passer les feux d'une configuration à l'autre.*

Mode jour

Dans ce mode, chaque feu passera, de façon habituelle, par une séquence vert, orange, rouge. Tant que le feu de la direction A est vert ou orange, celui de la direction B est rouge, et vice versa. L'ordinogramme correspondant à l'algorithme de contrôle apparaît figure 5-13. Les flèches, à droite de l'ordinogramme, indiquent la durée d'allumage de chaque feu. Si nous appelons D1 la durée du vert pour A, D2 la durée de l'orange pour A, D3 et D4 les durées du vert et de l'orange pour B, on voit, par inspection du diagramme, que la durée totale du cycle est

$$D1 + D2 + D3 + D4.$$

Dans la réalité, ces délai sont soumis à des contraintes. En particulier, le cycle d'un carrefour est habituellement compris entre 1 et 2 minutes. La plupart des conducteurs ne tolèrent pas, en effet, une durée du feu rouge supérieure à deux minutes : ils franchissent tout simplement le carrefour une fois leur patience épuisée, en supposant que le feu est en panne. Les durées des autres feux sont également soumises à des contraintes qui proviennent de la nécessité de respecter les délais nécessaires à un piéton ou à une

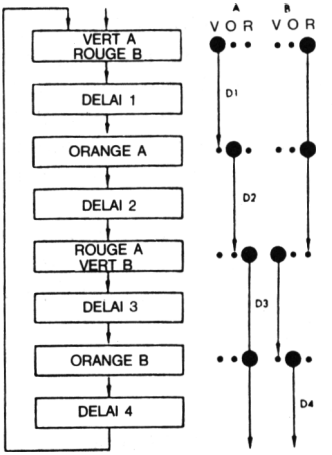


Fig. 5-13 : Mode jour (commandes d'extinction non représentées)

voiture, pour libérer l'intersection, une fois qu'ils y ont pénétré. La durée de l'orange est ainsi appelée « temps de libération » parce qu'elle représente le temps qu'il faut à une voiture pour libérer le carrefour. Le feu vert peut avoir n'importe quelle durée minimum dès lors qu'aucun piéton ne traverse le carrefour. Si les piétons sont autorisés à traverser, la durée minimum du feu rouge devra être telle qu'ils puissent libérer le carrefour en toute sécurité. Exemple : la durée du rouge dans la direction B est égale à D1 + D2. Si nous supposons que le minimum de l'orange, dans la direction A, est de 3 secondes, et que le minimum du rouge, dans la direction B est de 10 secondes, nous pouvons voir, sur la figure 5-13, que la durée minimum du vert, dans la direction A, est

D1 = 10 − 3 = 7 secondes.

Mathématiquement, si nous posons :

VERT A = D1
ORANGE A = D2
VERT B = D3
ORANGE B = D4

alors :

ROUGE A = D3 + D4
ROUGE B = D1 + D2

En général, le cycle est fixe, et :

D1 + D2 + D3 + D4 = CONSTANTE

Dans notre programme, nous utiliserons des cycles plus rapides que dans la réalité, simplement parce qu'il est exaspérant d'attendre une ou plusieurs minutes pour observer si le programme fonctionne correctement. Pour des raisons pratiques, il est souhaitable de s'en tenir pour les tests à un cycle de 10 à 20 secondes. Le lecteur doit maintenant avoir acquis les connaissances indispensables pour ajuster facilement les délais, de façon à pouvoir connecter son micro-ordinateur à un carrefour réel. Le programme apparaît figure 5-14.

```
0140  A9 3F      DAY    LDA  #$3F     Met DDRA du VIA n° 1 à $3F pour
0142  8D 03 A0           STA  $A003      mode sortie

0145  A9 21      ONDAY  LDA  #$21     Met une direction au vert et l'autre au
0147  8D 01 A0           STA  $A001      rouge

014A  A9 D0              LDA  #$D0     Met le compte pour DLYA (adresse
014C  85 00              STA  $00        0000) a la valeur $D0

014E  20 20 01           JSR  DLYA     Appelle la routine de délai
0151  A9 22              LDA  #$22     Allume orange et rouge
0153  8D 01 A0           STA  $A001
0156  A9 EA              LDA  #$EA     Met le compte pour DLYA à $EA
0158  85 00              STA  $00
015A  20 20 01           JSR  DLYA     Appelle le délai
015D  A9 0C              LDA  #$0C     Allume rouge et vert
015F  8D 01 A0           STA  $A001
0162  A9 D0              LDA  #$D0
0164  85 00              STA  $00      Met le compte pour DLYA à $D0
0166  20 20 01           JSR  DLYA     Appelle le délai
0169  A9 14              LDA  #$14     Allume rouge et orange
016B  8D 01 A0           STA  $A001
016E  A9 E8              LDA  #$E8
0170  85 00              STA  $00      Met le compte pour DLYA à $E8
0172  20 20 01           JSR  DLYA     Appelle le délai
0175  4C 45 01           JMP  ONDAY    Répète
```

Fig. 5-14 (Programme 5-2) : Simulation de feux de carrefour - Mode jour (Branchement : connecteur A au connecteur H1)

Comme dans le programme précédent, nous configurerons le registre direction DDRA en sortie, pour commander les 6 LED connectées au port A, grâce aux deux premières instructions du programme.

```
DAY     LDA  #$3F
        STA  $A003
```

Nous allumerons, ensuite, le vert, pour la direction A, et le rouge pour B, grâce aux deux instructions suivantes, qui chargent le motif binaire approprié (21 hexadécimal) dans le registre d'E/S :

```
ONDAY   LDA   #$21
        STA   $A001
```

La durée d'un délai sera spécifiée en déposant une valeur dans la case mémoire 00. On appelle alors le sous-programme de délai :

```
        LDA   #$D0
        STA   $00
        JSR   DLYA
```

Le processus sera répété pour l'orange direction A. Pour le rouge direction A associé au vert direction B, et enfin pour l'orange direction B. Puis, retour au départ :

```
        LDA   #$22      ORANGE A ROUGE B
        STA   $A001
        LDA   #$EA
        STA   $00
        JSR   DLYA      DÉLAI
        LDA   #$0C      ROUGE A VERT B
        STA   $A001
        LDA   #$D0
        STA   $00
        JSR   DLYA      DÉLAI
        LDA   #$14      ROUGE A ORANGE B
        STA   $A001
        LDA   #$E8
        STA   $00
        JSR   DLYA      DÉLAI
        JMP   ONDAY     ON RECOMMENCE
```

Le lecteur aura intérêt à vérifier que ce programme correspond exactement à l'ordinogramme de la figure 5-13, dont l'interprétation devrait maintenant être totalement évidente. Il est vivement conseillé d'essayer différentes constantes de temps, par rapport à

celles indiquées dans le programme, et de vérifier la conformité des temps. Envisageons maintenant différentes améliorations à cet algorithme de contrôle de carrefour.

Par exemple, on pourra modifier le programme de telle sorte que le temps de l'orange, le temps du rouge et celui du cycle, soient spécifiés par le temps pendant lequel un des interrupteurs est maintenu appuyé, juste après démarrage du programme.

Exercice 5-3 : *Implantez un algorithme à "réponse dynamique" : le temps du vert pour l'avenue A sera allongé de 5 secondes, jusqu'à concurrence de 3 minutes, chaque fois qu'on décèlera une requête à la « boucle détectrice » (simulée par un interrupteur).*

Exercice 5-4 : *Implantez des "appels piétons", à l'aide d'interrupteurs. On donnera le feu vert au piéton le plus vite possible tout en respectant les temps de libération minimum.*

Exercice 5-5 : *Implantez un "interrupteur de police". En appuyant sur l'un des interrupteurs, l'intersection passera, manuellement, par sa séquence. Deux pressions rapides permettent de revenir au fonctionnement automatique.*

LED EN MATRICE DE POINTS

Nous utiliserons ici un afficheur à LED disposées en matrice 5 × 7 (cf. fig. 5-15). On a recours à ce type de matrice dans un grand nombre d'applications. Par exemple, les imprimantes à matrice font souvent appel à une matrice de points 5 × 7, pour imprimer les caractères sur le papier. Les moniteurs télévision utilisent aussi une matrice de points, pour afficher les caractères sur l'écran. La matrice 5 × 7 est le minimum standard pour obtenir une représentation acceptable des caractères. La lisibilité n'est toutefois pas idéale. Il faut utiliser pour cela des matrices de points plus grandes, 7 × 9 par exemple, dont le prix est évidemment plus élevé. Dans cette application, nous connecterons directement une

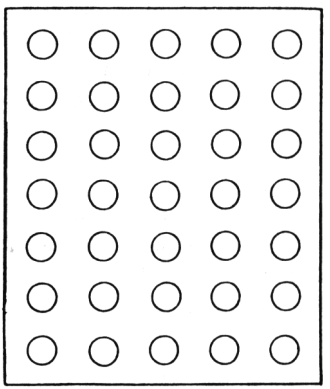


Fig. 5-15 : Led en matrice 5 × 7

matrice de LED 5 × 7 aux registres d'E/S B des 6522 n^{os} 1 et 3. Il faudrait normalement utiliser des amplificateurs, pour que les LED donnent assez d'intensité lumineuse. Ici, pour limiter le nombre de composants, nous connecterons les LED directement. Conséquence : les LED apparaîtront en sombre sur la carte et l'affichage sera assez difficile à lire. Pour améliorer le rendement, vous pouvez ajouter des amplis sur les lignes. Le branchement de la matrice de LED apparaît figure 5-16. Les 7 lignes numérotées de 1 à 7 sont reliées, respectivement, aux bits 7, 5, 4, 3, 2, 1 et 0 du registre d'E/S B du 6522 n° 1. Le bit 6 de ce registre IORB n'est pas disponible sur la carte SYM, car le moniteur le consacre à l'entrée cassette. Par suite, l'état de ce bit 6 sera indifférent.

Les cinq colonnes de l'afficheur à LED, numérotées de 1 à 5 sont, respectivement, reliées aux bits 0, 1, 2, 3, et 4 de IORB du 6522 n° 3.

On se reportera à la figure 5-16. Les deux registres IORB sont, respectivement, aux adresses A000 et AC00.

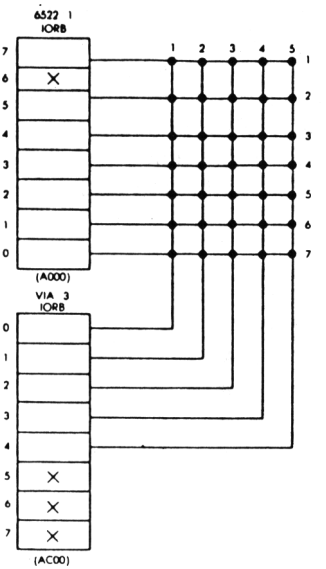


Fig. 5-16 : Connexion de la matrice de LED 5 × 7

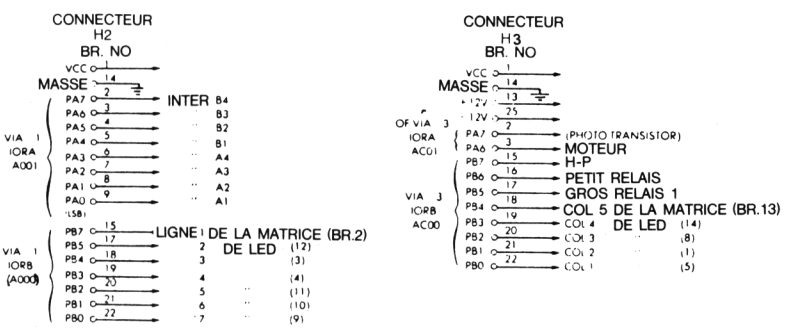


Fig. 5-17 : Les connecteurs pour les LED

Le problème fondamental est de sélectionner la combinaison convenable de lignes et de colonnes, pour éclairer les points représentant un caractère. Chaque caractère peut être représenté dans une matrice 5 × 7. Nous afficherons, par exemple, tous les caractères hexadécimaux, c'est-à-dire tous les chiffres de 0 à 9, et toutes les lettres de A à F. Examinons leur codage.

Une LED allumée sera représentée par un bit "0". Une LED éteinte par un bit "1". Une LED sera allumée en mettant sa ligne

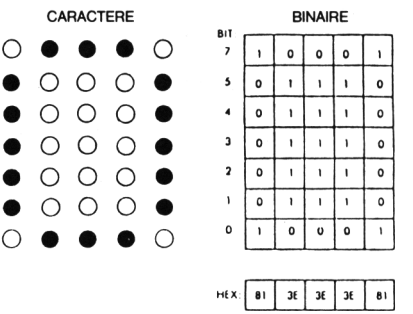


Fig. 5-18 : Affichage de « 0 »

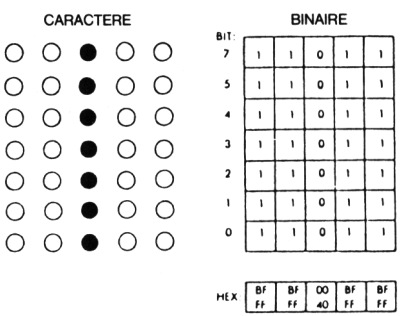


Fig. 5-19 : Affichage de « 1 »

à 0. Le motif approprié pour afficher un zéro apparaît figure 5-18. Naturellement, l'utilisateur est libre de choisir tout autre motif de son choix. A titre d'exercice, il pourra afficher le "0" avec des coins carrés, plutôt qu'arrondis. Il lui suffira de modifier la table en conséquence.

Le codage binaire correspondant apparaît à droite de la figure 5-18. L'équivalent hexadécimal est indiqué en bas de la table binaire. Rappellons que la ligne 6 est inutilisée. Elle est indifférente, c'est-à-dire qu'on peut la considérer, soit comme 0, soit comme 1, comme l'illustre le codage du "0" sur la figure 5-18. La première colonne a pour valeur "1000001" ou, plus exactement, "1 – 000001", où " – " représente la valeur indifférente du bit 6. Supposons que ce dernier soit mis à 0. La valeur de la première colonne est alors, "10000001" ("81" en hexadécimal).

De même, la valeur de la seconde colonne est, en ajoutant un 0 pour le bit 6 : "00111110" ("3E" en hexadécimal).

Pour le chiffre "0", les cinq colonnes sont donc :

81, 3E, 3E, 3E, 81.

Regardons maintenant le caractère "1" (fig. 5-19). Le codage binaire correct apparaît à droite de l'illustration.

En supposant que le bit 6 est à 0, les codes hexadécimaux correspondants sont :

BF, BF, 00, BF, BF

Si le bit 6 était à 1, les codes seraient :

FF, FF, 40, FF, FF

On peut utiliser n'importe quelle valeur 0 ou 1 pour le bit 6, dans n'importe quelle colonne, du moment que ce bit ne reçoit pas une autre affectation. La figure 5-21 donne la table complète du codage des caractères "0" à "F".

Exercice 5-7 : *Montrez la forme des caractères 0 à F qui résulte de cette table.*

Exercice 5-8 : *Réécrivez la table de façon plus cohérente, en mettant partout le bit 6 à "0".*

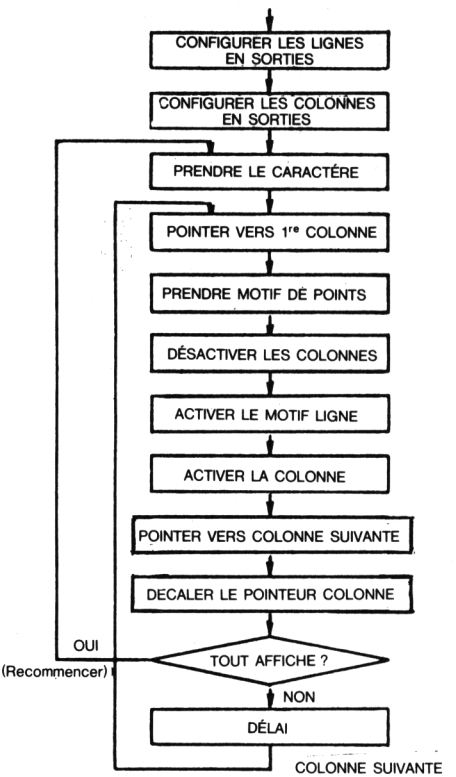


Fig. 5-20 : Commande de la LED en matrice de points

L'ordinogramme pour le programme de commande de la LED est listé figure 5-21. Les lignes et les colonnes sont configurées en sorties, en chargeant les motifs binaires appropriés dans les registres direction correspondants des 6522. On doit alors afficher le motif de points correspondant au caractère. Les points seront éclairés, successivement, pour chaque colonne de l'afficheur. Pour chaque caractère, le programme doit accéder à cinq entrées successives de

la table des matrices de points. Ces entrées correspondent aux cinq colonnes de points adaptées à chaque caractère. Le programme particulier va alors indéfiniment boucler et afficher le caractère. Les points seront éclairés en éteignant les colonnes (ce qui efface le motif précédent), puis en activant, à la fois, le motif de ligne correspondant aux points désirés, et la colonne où les points doivent apparaître. Ensuite, on passe à la colonne suivante. Tous les points doivent être éclairés pendant un temps égal, pour donner l'impression d'un éclairement uniforme. De plus, toutes les colonnes doivent être balayées pendant une période inférieure à 1/10 de seconde, pour éviter tout scintillement. La routine de délai à la fin du programme, est ajustée en conséquence. Le programme apparaît ci-dessous et à la page suivante.

Caractère	adr. basse	col 1	col 2	col 3	col 4	col 5
0	90	81	3E	3E	3E	81
1	95	FF	FF	00	FF	FF
2	9A	DE	7C	7A	76	CE
3	9F	DD	76	76	76	C9
4	A4	F3	EB	DB	00	FB
5	A9	05	76	76	76	79
6	AE	C1	76	76	76	D9
7	B3	7F	7F	7F	7F	00
8	B8	C9	76	76	76	C9
9	BD	CD	76	76	76	C1
A	C2	E0	DB	7B	DB	E0
B	C7	00	76	76	76	C9
C	CC	C1	7E	7E	7E	DD
D	D1	00	7E	7E	7E	C1
E	D6	00	76	76	76	76
F	DB	00	77	77	77	77

La table réside aux adresses 0090-00DF.

Fig. 5-21 : Table des matrices de points

Connexions : Connecteur A à H2
Connecteur AA à H3

Ce programme prend en compte les 8 bits les moins significatifs, de l'adresse du caractère à l'adresse 0001, puis se rend à la table de la figure 5-21, pour prendre le motif correspondant au caractère choisi, qu'il affiche alors sur la matrice de LED.

Avant de passer à l'exécution vous chargerez en 0001, l'octet bas de l'adresse du caractère. L'octet le plus significatif de l'adresse du caractère est 00 (page 0).

La table des caractères doit être rangée en page 0 comme indiqué figure 5-21.

Fig. 5-22 : Programme d'affichage sur matrice de LED (Programme 5-3)

Notes :

1° On peut utiliser un générateur de caractères à la place de la table.

2° La matrice utilisée est 5 × 7 : 7 bits sont nécessaires pour définir le motif d'une colonne, mais la table en utilise 8. La raison en est que le programme utilise le registre IORB du VIA n° 1 pour commander les 7 lignes, et que, seuls, 7 bits de ce registre sont disponibles. Le bit 6 est indifférent, parce qu'il est dédié à l'entrée cassette.

```
0180  A9 BF      BSCLED LDA  #$BF     Avant exécution, on doit mettre dans
                                      0001 l'adresse du caractère choisi
0182  8D 02 A0          STA  $A002    $BF en DDRB du VIA n° 1 pour
0185  A9 1F              LDA  #$1F    commander 7 lignes

0187  8D 02 AC          STA  $AC02    $1F en DDRB du VIA n° 3 pour
                                      commander 5 colonnes
018A  A9 00              LDA  #$00    Mise à 0 de la partie haute de l'adresse
018C  85 03              STA  $03     du caractère choisi (dans la case
                                      mémoire 0003)
018E  A2 00              LDX  #$00
0190  A5 01      RPTCHA  LDA  $01     Déplacement de 0001 à 0002 de l'adresse
                                      basse du caractère à afficher
0192  85 02              STA  $02     (Y) = $10 pour activer la dernière
0194  A0 10              LDY  #$10    colonne

0196  A1 02      NXTCOL  LDA  $02     (A) = motif de la colonne courante du
                                      caractère à afficher
0198  8E 00 AC          STX  $AC00   Désactive toutes les colonnes avant
                                      d'activer les lignes
019B  8D 00 A0          STA  $A000   Active les lignes
019E  8C 00 AC          STY  $AC00   Active la colonne courante
01A1  E6 02              INC  $02     Avance l'adresse (en $0002) pour passer
                                      à la colonne suivante
01A3  98                 TYA
01A4  4A                 LSR  A       Décale (Y) de 1 bit vers la droite pour
                                      activer la colonne suivante
01A5  A8                 TAY
01A6  C0 00              CPY  #$00    (Y) = 00 signifie qu'on a affiché
                                      les 5 colonnes
01A8  D0 03              Sinon, branchement à DLY3 pour
                         compenser le temps (cf. note 1)

01AA  4C 90 01           JMP  RPTCHA  Si oui, on répète le caractère complet
01AD  A2 FF      DLY3    LDX  #$FF    Délai
01AF  E8         LP3     INX
01B0  EO 00              CPX  #$00
01B2  30 FB              BMI  LP3
01B4  4C 96 01           JMP  NXTCOL  Va activer la colonne suivante
```

Fig. 5-22 : Programme d'affichage sur matrice de LED (Programme 5-3) (suite)

Notes :

1° Une compensation est nécessaire, faute de quoi la dernière colonne serait toujours activée plus longuement : elle serait alors plus brillante que les 4 autres.

2° La compensation évoquée ci-dessus ne résout le problème que partiellement. La brillance est encore inégale, parce que le nombre de LED éclairées est différent pour chaque colonne. Un programme plus détaillé tiendrait compte du nombre de LED activées dans chaque colonne, pour effectuer la compensation de temps.

Fig. 5-22 (suite)

Les quatre premières instructions du programme conditionnent les registres direction pour les lignes et les colonnes. Ils sont mis en sortie :

```
BSCLED   LDA   #$BF
         STA   $A002      VIA n° 1 : 7 LIGNES
         LDA   #$1F
         STA   $AC02      VIA n° 2 : 5 COLONNES
```

Par convention, l'emplacement de la table du caractère à afficher sera ici contenu à l'adresse "01", en page 0. Cet emplacement est, par exemple, 90 pour le caractère "0", 95 pour le caractère "1", et ainsi de suite, comme indiqué dans la table au début du programme. (On suggérera plus bas une amélioration.) Pour afficher le caractère "2", il faudra auparavant avoir déposé la valeur 9A dans la case d'adresse "01". Nous aurons à pointer successivement, sur 5 éléments de la table, pour chaque colonne du caractère. Il sera donc indispensable de générer les adresses 9A, 9B, 9C, 9D et 9E. Pour ne pas détruire notre pointeur de départ "9A", nous utiliserons deux cases mémoire supplémentaires, aux adresses 02 et 03, pour contenir le pointeur courant à la colonne en cours d'affichage. Dans la mesure où nous opérons en page 0, le contenu de l'adresse 03 sera mis à 0 (octet haut de l'adresse).

```
LDA   #$00
STA   $03
```

Quand nous entrerons dans la boucle principale, le registre X sera supposé contenir 0. Il sera utilisé pour désactiver un registre de sorties.

```
LDX   #$00
```

Première colonne soumise au pointage : celle qui se trouve à l'adresse spécifiée dans la case 01 (pointeur vers la table caractères). Nous transférerons le contenu de la case mémoire 01 dans 02 :

```
RPTCHA   LDA   $01
         STA   $02
```

Le registre Y sera utilisé comme compteur à décalages et servira, en même temps, à activer sélectivement une colonne. Il sera mis, initialement, à la valeur "10", pour activer la première colonne :

```
         LDY   #$10
```

Le "1" sera ensuite décalé d'un cran vers la droite, pour activer la colonne suivante, et ainsi de suite. Quand le "1" tombera finalement hors du registre, les 5 colonnes du caractère auront été éclairées. La boucle pourra être répétée. Ce registre n'étant pas seulement utilisé pour activer une colonne, mais aussi pour compter jusqu'à 5, il sera appelé *compteur à décalages*. Le motif de points de la colonne courante est obtenu en accédant à l'élément de la table dont l'adresse est contenue en 02. Soit, comme le contenu de X est 00 :

```
NXTCOL   LDA   ($02, X)
```

Le motif de points est maintenant contenu dans l'accumulateur. Affichons-le. On désactivera d'abord toutes les colonnes en envoyant 0 dans IORB (PIO n° 3) :

```
         STX   $AC00
```

Avant d'envoyer le contenu de l'accumulateur dans IORB (PIO n° 1) pour activer les lignes :

```
         STA   $A000
```

On activera enfin la colonne appropriée, pour provoquer l'éclairage des LED sélectionnées :

```
         STY   $AC00
```

Une LED ne s'éclairera que si elle est connectée à une colonne active, et à une ligne mise à 0. Chaque "0" dans le motif de points va éclairer le point correspondant dans la colonne sélectionnée.

On incrémentera la case mémoire "02", pour pointer vers le prochain motif de points du caractère considéré. Nous devrons alors décaler notre pointeur de colonne d'un cran à droite, et déterminer si toutes les colonnes ont déjà été éclairées.

```
INC   $02
TYA                 Y NE PEUT PAS ÊTRE
LSR   A             DÉCALÉ DIRECTEMENT
TAY
CPY   #$00
BNE   DLY3
JMP   RPTCHA
```

Il n'est pas possible de décaler le registre Y directement : on le transférera dans l'accumulateur, qui sera, lui, décalé, et dont le contenu sera recopié dans le registre Y. On comparera le contenu de Y à la valeur "0" (on peut suggérer une amélioration sur ce point). Si le contenu est 0, l'opération est terminée : les 5 colonnes ont été éclairées. Si tel n'est pas le cas, il convient de marquer un délai pendant lequel les LED resteront éclairées, avant de passer à la colonne suivante :

```
DLY3    LDX   #$FF
LP3     INX
        CPX   #$00
        BMI   LP3
        JMP   NXTCOL
```

Le registre X est utilisé comme compteur. On obtiendra un délai traditionnel en incrémentant le registre index, un nombre de fois déterminé, puis, on se branchera à l'adresse NXTCOL pour traiter la colonne suivante.

Améliorations du programme : Nous envisagerons des modifications d'écriture permettant de réduire le nombre d'instructions, puis apporterons des améliorations aux fonctions effectuées.

Exercice 5-9 : *Réécrire la routine de délai DLY3 de façon qu'elle occupe moins d'instructions.*

Exercice 5-10 : *Examinez les trois dernières instructions de la routine NXTCOL, à partir de l'adresse 01A6 (cf. fig. 5-22). Pouvez-vous suggérer une autre manière de tester si le dernier 1 a été sorti de Y ?*

Exercice 5-11 : *Ajoutez une routine à ce programme, pour éviter de déposer, à l'adresse 01, un pointeur vers l'élément de la table. On se bornera à déposer la valeur du caractère. Avec cette routine, l'utilisateur devra être capable de déposer une valeur comprise entre "0" et "F", et de faire en sorte qu'elle soit affichée correctement. Pour parvenir à ce résultat, il est nécessaire de convertir la valeur du caractère en adresse dans la table. Par exemple, "0" correspondra à "90" (cf. la table au début du programme 5-3), "1", correspondra à "95", etc. L'équation est : adresse de départ = 90 + code × 5.*

Note : Au lieu d'effectuer mot à mot la multiplication par 5, on peut utiliser un raccourci : rappelez-vous que décaler d'un cran à gauche revient à multiplier par 2, et que $5 = 2 \times 2 + 1$. Une multiplication par 4 peut être effectuée à l'aide de deux décalages à gauche successifs.

Exercice 5-12 : *Ecrivez une routine supplémentaire pour afficher une chaîne de caractères. Cette routine supposera que l'adresse de départ de la chaîne soit contenue dans la case mémoire d'adresse 01. Chaque caractère sera affiché pendant une seconde. La chaîne peut être terminée par n'importe quel caractère non compris entre 0 et F. Le programme fera alors une pause de deux secondes, et affichera à nouveau la chaîne.*

Considérons maintenant d'autres améliorations aux fonctions du programme. Nous allons ajouter quatre interrupteurs, et développer un programme qui affichera la valeur hexadécimale formée par les interrupteurs.

AFFICHAGE DES VALEURS FORMÉES PAR DES INTERRUPTEURS

Nous lirons, ici, les valeurs binaires de quatre interrupteurs, et afficherons le caractère hexadécimal correspondant sur la matrice de LED. L'ordinogramme de l'algorithme apparaît figure 5-23. Le programme va lire les quatre interrupteurs, pointer vers le début de la table de conversion définie au programme précédent, puis calculer pour le caractère à afficher le déplacement dans la table. L'adresse dans la table du code binaire correspondant aux points à éclairer est obtenue en multipliant la valeur du caractère par 5.

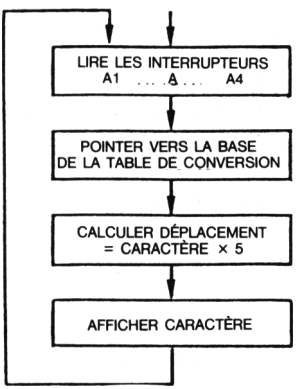


Fig. 5-23 : Affichage de la valeur des interrupteurs

Il est possible de le vérifier en examinant la table de la figure 5-22. Nous calculerons l'adresse de la première colonne à afficher, et la déposerons à l'adresse mémoire 01. Le programme précédent sera utilisé pour afficher le caractère sur les LED. Voici le programme :

Branchement : Connecteur A sur H2
Connecteur AA sur H3

Ce programme lit les interrupteurs A1 à A4 pour calculer l'une des 16 valeurs hexadécimales possibles, et l'afficher.

Il utilise le programme 5-3 comme sous-programme. Avant exécution il est utile d'effectuer les modifications suivantes :

1° A l'adresse 01AA, la valeur 4C doit être changée en 60 (60 est le code hexadécimal de RTS).

2° La constante de compensation de temps, à l'adresse 1AE est FF ; elle doit être changée en F0, parce que le présent programme active la dernière colonne plus longtemps que le programme 5-3.

Fig. 5-24 : Programme d'affichage sur matrice de LED perfectionné (programme 5-4)

```
0200  A9 00     RDCHA  LDA  #$00     Met DDRA du VIA n° 1 à 0 pour mode
0202  8D 03 A0         STA  $A003       entrée
                                       mode
0205  AD 01 A0         LDA  $A001     Lit les interrupteurs B1-B4 et A1-A4
0208  29 0F            AND  #$0F      Ignore B1-B4
020A  A8               TAY           Range l'état de A1-A4 en Y
020B  A2 90            LDX  #$90      Calcule l'adresse du caractère et la range
                                       en 0001 ; 90 est l'adresse de base
020D  86 01            STX  $01
020F  A2 00            LDX  #$00      Compteur d'additions
0211  18        ADD    CLC           A contient l'état des interrupteurs
0212  65 01            ADC  $01       Boucle cinq fois sur l'addition
0214  85 01            STA  $01       90 + (A)
0216  98               TYA
0217  E8               INX           Restaure l'état des interrupteurs en A
0218  E0 05            CPX  #$05      (X) = 5 veut dire calcul terminé
021A  30 F5            BMI  ADD
021C  20 80 01         JSR  BSCLED    Alors on appelle BSCLED pour afficher
021F  4C 00 02         JMP  RDCHA     Puis on relit les interrupteurs
```

Fig. 5-24 : (suite)

Le programme apparaît figure 5-24. Les deux premières instructions configurent le registre direction du port A en entrée, pour lire les interrupteurs :

```
RDCHA   LDA   #$00
        STA   $A003
```

On poursuivra par la lecture de l'état des interrupteurs A1 à A4, en ignorant l'état des interrupteurs B1 à B4 :

```
        LDA   $A001
        AND   #$0F        MASQUE B1-B4
```

L'état des interrupteurs est sauvé dans le registre index Y :

```
        TAY
```

On rangera l'adresse de départ de la table (90) à l'adresse 01 :

```
        LDX   #$90
        STY   $01
```

A cette adresse de départ, on ajoutera le déplacement approprié, pour accéder à la première colonne de points, correspondant au caractère spécifié par les interrupteurs. Le déplacement est calculé en multipliant la valeur des interrupteurs par 5. On utilisera le registre X, initialisé à 0, comme compteur de 0 à 5 :

```
        LDX   #$00
```

La valeur des interrupteurs est ajoutée au contenu de la case-mémoire 01 :

```
ADD     CLC
        ADC   $01
        STA   $01
```

L'instruction CLC (mettre retenue à 0) doit être utilisée avant toute addition. En outre, nous supposerons que le mode binaire a été établi d'avance (le 6502 peut fonctionner soit en mode binaire, soit en mode décimal). Si aucune autre condition n'est imposée, le 6502 fonctionnera normalement, en mode binaire, puisqu'un reset remet à 0 le registre d'état.

On restaurera la valeur des interrupteurs dans l'accumulateur, à partir du registre index, où elle a été sauvegardée. Le compteur d'additions est incrémenté et comparé à 5 :

```
        TYA
        INX
        CPX   #$5
        BMI   ADD
```

On répète l'addition tant que la valeur 5 n'a pas été atteinte. Après la case-mémoire reçoit la bonne valeur, et on appelle le sous-programme BSCLED (le programme d'affichage précédent) :

```
        JSR   BSCLED
```

Pour finir, le programme reboucle, afin de relire les interrupteurs, et d'afficher le caractère qu'ils spécifient :

```
        JMP   RDCHA
```

PRODUCTION D'UN SON

Nous avons vu, au chapitre précédent, la procédure à suivre pour générer un son, en envoyant, simplement, un signal à la fréquence désirée sur un haut-parleur. Le signal carré est produit en envoyant, alternativement, 0 et 1 sur le haut-parleur. Le temps pendant lequel le haut-parleur est à 0 ou à 1 s'appelle la demi-période. La mesure du délai peut s'effectuer aussi bien par software que par hardware, à condition de faire appel, dans ce dernier cas, au générateur d'intervalles de temps incorporé dans le 6522. Ce temporisateur a déjà été utilisé auparavant : notre préférence ira donc ici à une méthode software. Nous développerons, d'abord, un programme élémentaire capable de produire un son, puis nous l'améliorons, de façon à créer de la musique par ordinateur.

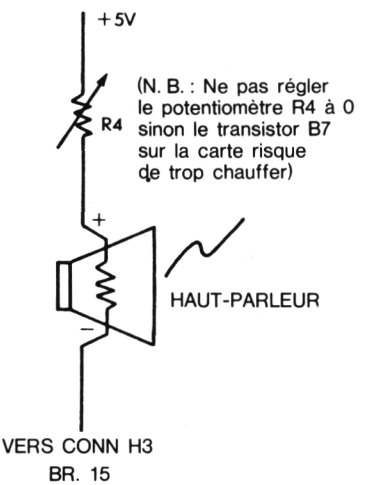


Fig. 5-25 : Branchement du haut-parleur

Le branchement hardware est décrit par la figure 5-25. Il serait bon de placer une résistance supplémentaire d'au moins 50 ohms en série avec le haut-parleur, pour limiter le courant de sortie. Le haut-parleur est relié à une sortie amplifiée du SYM. Si le potentiomètre est positionné à la valeur 0, le transistor, qui se trouve sur la carte, peut être détruit.

La technique employée pour produire un son est la méthode habituelle aux signaux carrés, réalisée à l'aide d'un sous-programme de délai.

Branchement : Connecteur A au connecteur H2
Connecteur AA au connecteur H3

Ce programme excite le haut-parleur à une fréquence pré-établie qui doit être chargée dans la case-mémoire 0004 avant exécution.

```
0230  A9 80     BSCSPK LDA  #$80     Met DDRA du VIA n° 3 à $80 pour
0232  8D 02 AC         STA  $AC02     que le bit 7 soit en sortie sur le haut-
                                       parleur
0235  A9 80     AGAIN  LDA  #$80     Met à 1 le bit du haut-parleur
0237  8D 00 AC         STA  $AC00

023A  20 48 02         JSR  DLYB     Appelle le délai
023D  A9 00            LDA  #$00     Met à 0 le bit du haut-parleur
023F  8D 00 AC         STA  $AC00

0242  20 48 02         JSR  DLYB     Appelle le délai
0245  4C 30 02         JMP  AGAIN    Recommence
```

Sous-programme DLYB : Ce sous-programme est analogue à DLYA à part que :
1° Le délai est beaucoup plus court
2° On prend la valeur du délai à l'adresse 0004 (la valeur doit être négative)

```
0248  A6 04     DLYB   LDX  $04      Charge la valeur du délai en X
024A  E8        LPXB   INX           Incrémente X
024B  E0 00            CPX  #$00
024D  30 FB            BMI  LPXB     Boucle jusqu'à ce que (X) = 0
024F  60               RTS
```

Fig. 5-26 : Programme de base d'activation d'un haut-parleur (Programme 5-5)

Le paramètre de délai doit être chargé à l'adresse 0004, avant exécution. Il contrôlera la fréquence du son produit. La figure 5-26 décrit le programme. Le bit 7 du registre direction du port B est mis en sortie :

```
BSCSPK    LDA   #$80
          STA   $AC02
```

Le haut-parleur est mis à 1 :

```
AGAIN     LDA   #$80
          STA   $AC00
```

Il est laissé dans cette position pendant un temps spécifié par le contenu de l'adresse-mémoire 0004, en appelant le sous-programme DLYB :

```
        JSR     DLYB
```

Le haut-parleur est ensuite mis à 0 en même temps que le bit 7 de IORB :

```
        LDA     #$00
        STA     $AC00
```

Il sera maintenu dans cette position pendant la même durée par l'intermédiaire d'un appel à DLYB :

```
        JSR     DLYB
```

Le programme boucle sur lui-même :

```
        JMP     AGAIN
```

Le sous-programme de délai est, pour l'essentiel, identique au sous-programme DLYA du programme 5-1 :

```
DLYB    LDX     $04         VALEUR DU DÉLAI
LPXB    INX                 COMPTEUR
        CPX     #$00
        BMI     LPXB
        RTS
```

Calculons la durée du délai introduit par ce sous-programme. La durée de chaque instruction est indiquée, ci-dessous, à droite de l'instruction :

```
                            Nombre de cycles
        LDX     $04             (2)
        INX                     (2)
        CPX     #$00            (2)
        BMI     LPXB            (3)
        RTS                     (6)
```

L'instruction JSR (appel de sous-programme) utilisée introduit un délai de 6 cycles. La boucle est utilisée 256 − 128 = 128 fois, si la constante écrite en 004 est 80. Le délai total est donc :

$$6 + 2 + (2 + 2 + 3) \times 128 + 6 = 14 + 7 \times 128 = 910\ \mu s$$

Exercice 5-13 : *Modifier la routine de délai de manière à utiliser une instruction de décrémentation, plutôt que d'incrémentation.*

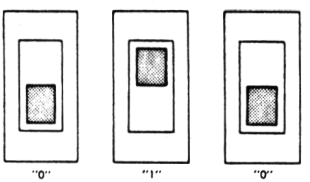


Fig. 5-27 : Utilisation d'interrupteurs pour spécifier la note

MUSIQUE

Après avoir présenté la méthode de base capable de générer un son de fréquence déterminée, nous allons maintenant nous efforcer de jouer un air. Le programme va lire la valeur binaire des trois interrupteurs A-1 à A-3, et produire un son correspondant à l'état des interrupteurs (cf. fig. 5-27). La note "do" sera produite pour la valeur "0" des interrupteurs, "ré" pour la valeur "1", etc. Il est possible de jouer toute une octave plus une note (c'est-à-dire de "do" à "do"), avec trois interrupteurs. Ce programme utilisera le précédent comme sous-programme. Avant de passer à l'exécution il est nécessaire de changer le contenu de la case-mémoire : 0245 ("4C"), en "60" (RTS). Nous construirons une table des fréquences, qui spécifiera la demi-période du signal carré correspondant à chaque note. Elle apparaît figure 5-28.

```
0050  A2 80      TUNE  LDX  #$80   Fréquence pour ut 3
0052  4C 74 02         JMP  LD04
0055  A2 90            LDX  #$90   Fréquence pour ré
0057  4C 74 02         JMP  LD04
005A  A2 9C            LDX  #$9C   Fréquence pour mi
005C  4C 74 02         JMP  LD04
005F  A2 A4            LDX  #$A4   Fréquence pour fa
0061  4C 74 02         JMP  LD04
0064  A2 B0            LDX  #$B0   Fréquence pour sol
0066  4C 74 02         JMP  LD04
0069  A2 B8            LDX  #$B8   Fréquence pour la
006B  4C 74 02         JMP  LD04
006E  A2 C0            LDX  #$C0   Fréquence pour si
0070  4C 74 02         JMP  LD04
0073  A2 C4            LDX  #$C4   Fréquence pour do
0075  4C 74 02         JMP  LD04
```

Fig. 5-28 : Table des fréquences pour la musique

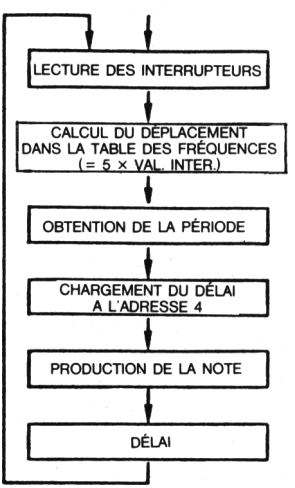


Fig. 5-29 : Ordinogramme du programme de musique

Branchement : Connecteur A au connecteur H2
Connecteur AA au connecteur H3

Ce programme lit les interrupteurs A1-A3 et active le haut-parleur avec la fréquence déterminée parmi 8 par les interrupteurs.

Il utilise le programme n° 5 comme sous-programme. Il en résulte qu'avant exécution, la valeur 4 C à l'adresse 0245 doit être changée en 60.

```
0250  A9 00      MUSIC  LDA  #$00     Précharge la partie haute de l'adresse de
                                      saut indirect
0252  85 05             STA  $05      A l'adresse 0005 ( = 00 car la table des
                                      fréquences est en page 0)
0254  8D 03 A0          STA  $A003    Met le DDRA du VIA n° 1 à 0 pour
                                      mode entrée
0257  A0 C0      KEY    LDY  #$C0     (Y) = constante de durée pour chaque fréquence
0259  AD 01 A0          LDA  #$A001   Lit l'état des interrupteurs
025C  29 07             AND  #$07     Ignore les 5 bits les plus significatifs
025E  85 04             STA  $04      Sauve l'état des interrupteurs en $04
0260  18                 CLC
0261  65 04             ADC  $04
0263  65 04             ADC  $04
0265  65 04             ADC  $04
0267  65 04             ADC  $04      Calcule l'adresse relative dans la table des
                                      fréquences
0269  85 04             STA  $04
026B  A9 50             LDA  TUNE     Ajoute l'adresse de base de la table des
                                      fréquences
026D  65 04             ADC  $04
026F  85 04             STA  $04      Range l'adresse calculée (partie basse) en
                                      $0004
0271  6C 04 00          JMP  ($0004)  Saut indirect dans la table des fréquences
0274  86 04      LD04   STX  $04      Prend la fréquence voulue
0276  20 30 02   CBSPK  JSR  BSCSPK   Appelle BSCSPK pour activer le haut-
0279  88                 DEY           parleur
027A  C0 00             CPY  #$00     Boucle jusqu'à ce que (Y) = 0 avant de
                                      relire les interrupteurs
027C  D0 F8             BNE  CBSPK    Boucle
027E  4C 57 02          JMP  KEY      Va relire les interrupteurs
```

Fig. 5-30 : Le programme de musique (Programme 5-6)

L'ordinogramme du programme apparaît figure 5-29. Le programme a pour tâche de lire le contenu des trois interrupteurs, puis de calculer le déplacement nécessaire pour obtenir le délai correspondant dans la table des fréquences.

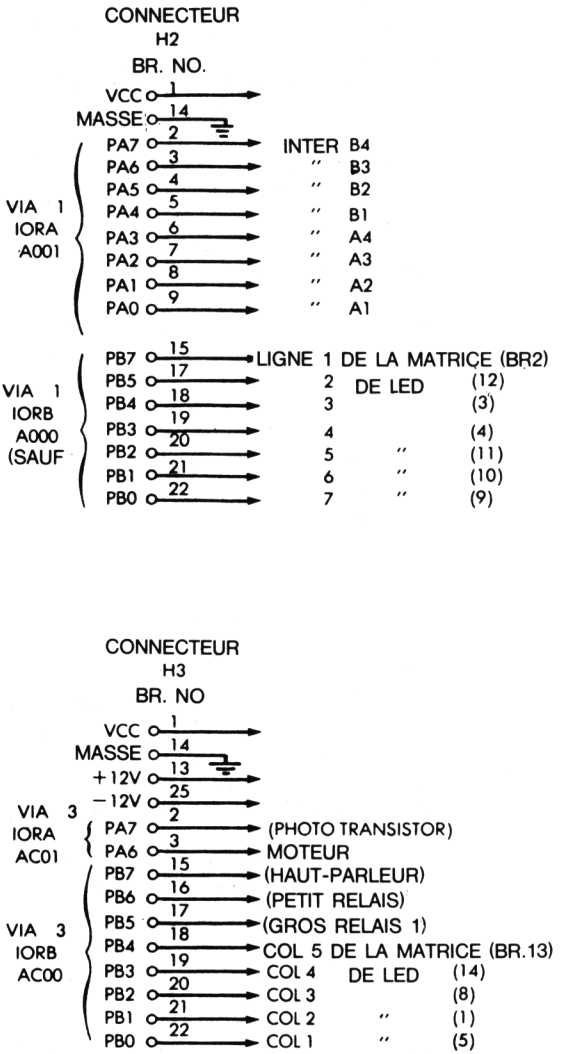


Fig. 5-31 : Connexions pour le programme de musique

Ce déplacement est égal à 5 fois la valeur formée par les interrupteurs. On obtient alors la période du signal carré. La note est produite pendant un temps donné. Le programme boucle sur lui-même, pour pouvoir jouer la note suivante (ou la même). La figure 5-30 montre le programme, et la figure 5-31 les connexions à réaliser. On utilisera les adresses 04 et 05 pour un saut indirect. Comme la table des fréquences réside en page 0, le contenu de 05 sera immédiatement mis à 0 :

```
MUSIC   LDA   #$00
        STA   $05
```

On mettra également à 0 le registre direction DDRA, pour imposer le mode entrée :

```
        STA   $A003
```

La durée du son est spécifiée par le registre Y, qui correspond à une boucle externe (expliquée plus bas) :

```
KEY     LDY   #$C0
```

On lira, ensuite, l'état des trois interrupteurs A1, A2 et A3, qui se trouvent dans l'IORA à l'adresse A001, et on masquera (mettra à 0) les 5 bits les plus à gauche :

```
        LDA   $A001
        AND   #$07
```

Cet état des interrupteurs est sauvegardé en mémoire, à l'adresse 04, pour libérer l'accumulateur en vue d'une autre utilisation :

```
        STA   $04
```

Pour calculer le déplacement dans la table des fréquences, il convient de multiplier par 5 la valeur obtenue, à partir des interrupteurs. Dans le cas présent, on ajoutera cette valeur 4 fois à elle-même :

```
        ADC   $04
        ADC   $04
        ADC   $04
        ADC   $04
        STA   $04
```

La valeur qui résulte du déplacement est rangée à l'adresse-mémoire 04. Nous sommes maintenant prêts à obtenir la demi-période dans la table des fréquences :

```
LDA   TUNE      ADRESSE DE BASE
ADC   $04
STA   $04       BASE + DÉPLACEMENT
JMP   ($0004)   SAUT INDIRECT
STX   $04       CONSTANTE
                DE LA FRÉQUENCE
```

La valeur est renvoyée dans le registre X, et sauvegardée en mémoire, à l'adresse 04. On appelle le sous-programme BSCSPK pour actionner le haut-parleur :

```
CBSPK   JSR   BSCSPK
```

Et cela autant de fois que le spécifie le contenu du registre Y :

```
DEY
CPY   #$00
BNE   CBSPK
```

On relira les interrupteurs, lorsque le son aura été produit pendant la durée voulue :

```
JMP   KEY
```

Améliorons ce programme :

Exercice 5-14 : *Nous pourrions simplifier la table des fréquences, en ne rangeant que la valeur du délai, c'est-à-dire $80, $90, etc. Vous modifierez le programme ci-dessus, de façon à utiliser la valeur des interrupteurs comme indice permettant de retrouver un élément dans cette nouvelle table. A noter, le raccourcissement considérable du programme entraîné par cette modification.*

Exercice 5-15 : *En faisant réellement tourner ce programme sur un micro-ordinateur, vous rencontrerez un problème mineur : le programme joue bien la note voulue, mais simultanément il produit une note de fréquence plus basse. En examinant avec soin les 5 dernières instructions du programme de musique, vous devriez pouvoir déterminer la nature du problème. Pouvez-vous proposer une modification du programme qui l'élimine ? (Conseil : le haut-parleur est peut-être mis à 0 trop longtemps.)*

Exercice 5-16 : *Regardez l'instruction "ADC $04", répétée 4 fois, et suggérez une manière d'obtenir le même résultat avec, si possible, moins d'instructions.*

Exercice 5-17 : *La troisième instruction à partir de la fin : "CPY # $00", est-elle nécessaire ?*

La table utilisée dans le programme de musique a été établie « à l'oreille », et non en calculant les fréquences correctes. Il faudrait maintenant vérifier les valeurs, pour évaluer sa qualité.

En France, la fréquence standard est la = 440 Hz. La fréquence double tous les 12 demi-tons. De la note 1 à la note 2, la fréquence est de : $N_2 = \sqrt[12]{2} \times N_1$.

Les fréquences sont données par la figure 5-28.

Exercice 5-18 : *Examinez la routine BSCSPK pour déterminer sa durée. Connaissant les périodes des notes (fig. 5-28), calculez les fréquences théoriques correctes. (N'oubliez pas que le haut-parleur est alternativement à 0 et à 1, pendant une demi-période.)*

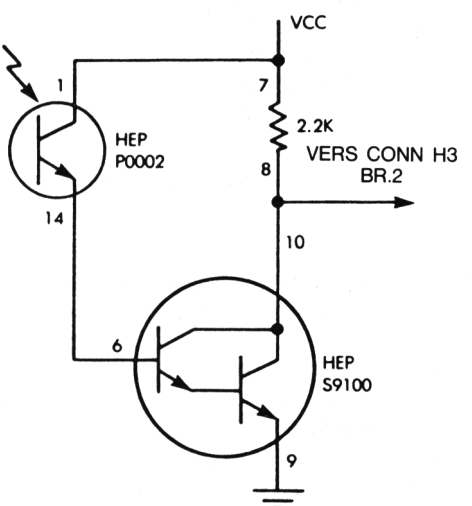


Fig. 5-32 : Le circuit du photo-transistor (sur le support M3)

ALARME ANTIVOL

Nouvelle application proposée : un système d'alarme domestique réaliste. Toute entrée dans la maison sera détectée par un phototransistor. On supposera que l'émetteur de lumière est, normalement, sous tension. Quand le faisceau lumineux est interrompu, le détecteur l'indique et l'alarme est déclenchée, sous forme d'un son de sirène dans le haut-parleur. Nous suggérerons des améliorations à la fin du programme.

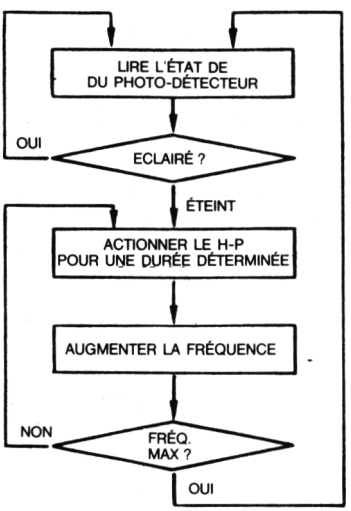


Fig. 5-33 : Ordinogramme de l'alarme

La figure 5-32 montre le branchement du phototransistor. L'ordinogramme est donné par la figure 5-33. Lisons l'état du photo-détecteur. S'il reste éclairé, cela veut dire que personne n'a interrompu le faisceau lumineux. On continue donc à tester. Lorsqu'une interruption a lieu, le détecteur est à 0, et le haut-parleur va être activé pendant un certain temps. Pour générer un son de sirène, il faut augmenter progressivement la fréquence du son, jusqu'à une fréquence maximum. Lorsqu'elle est atteinte (cf. fig. 5-34) , on teste à nouveau l'état du détecteur. La sirène retentit tant qu'il est maintenu à 0. Le programme apparaît figure 5-35. La sortie du phototransistor est reliée à l'entrée du bit 7 du registre IORA du VIA n° 3 (cf. fig. 5-32).

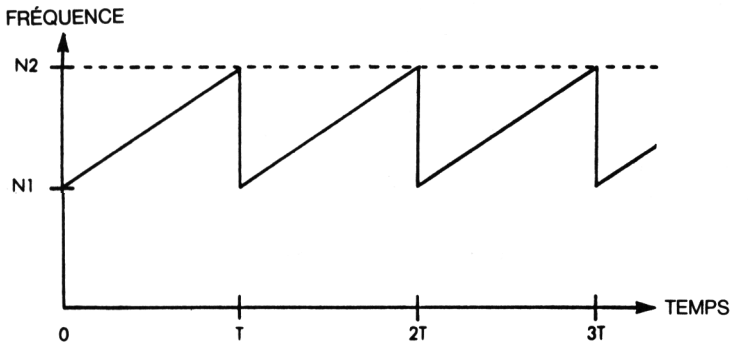


Fig. 5-34 : Son d'une sirène

Branchement : Connecteur A sur H2
Connecteur AA sur H3

Ce programme teste la sortie du phototransistor. Si elle est à 1, le transistor est éclairé, et rien n'arrive. Si elle est à 0, l'alarme est immédiatement déclenchée.

BSCSPK est, à nouveau, utilisé comme sous-programme. Conséquence : le contenu de l'adresse 0245 doit être changé en 60.

```
0281  A9 00      ALARM  LDA  #$00      Met DDRA du VIA n° 3 à 0 pour mode
0283  8D 03 AC          STA  $AC03       entrée

0286  AD 01 AC  DETECT LDA  $AC01      Lit la sortie du photo-transistor
0289  29 80             AND  #$80
028B  C9 80             CMP  #$80
028D  F0 F7             BEQ  DETECT    Si sortie à 1, continue à scruter
028F  A9 80             LDA  #$80      Sinon, sonne l'alarme en mettant à $80 la
                                         constante de fréquence initiale (à
0291  85 04             STA  $04         l'adresse 0004)
0293  A0 F0       LP7   LDY  #$F0      Met la constante de délai $F0 en Y
0295  20 30 02    SOUND JSR  BSCSPK    Appelle BSCSPK pour activer le haut-
0298  C8              INY                parleur
0299  C0 00             CPY  #$00      Boucle jusqu'à ce que (Y) = 0 avant de
029B  30 F8             BMI  SOUND       changer la fréquence
```

Fig. 5-35 : Alarme antivol (Programme 5-7)

```
029D  A9 01      LDA  #$01     Augmente de 1 la constante de fréquence
029F  18         CLC
02A0  65 04      ADC  $04
02A2  85 04      STA  $04
02A4  C9 A8      CMP  #$A8     Boucle jusqu'à ce que la constante de
                               fréquence la plus élevée soit égale à
02A6  30 EB      BMI  LP7      A8
02A8  4C 86 02   JMP  DETECT   Retourne tester le phototransistor
```

Fig. 5-35 : Alarme antivol (suite)

Les premières instructions du programme constituent une boucle de scrutation, qui teste l'état du phototransistor.

```
ALARM    LDA   #$00
         STA   $AC03
DETECT   LDA   $AC01
         CMP   #$80
         BEQ   DETECT
```

Dès que le photodétecteur est à 0 (1), on sonne l'alarme. La fréquence initiale spécifiée est chargée à l'adresse mémoire 04, et la durée du son de cette fréquence dans le registre Y. Le sous-programme précédent, BSCSPK, est appelé pour exciter le haut-parleur :

```
         LDA   #$80
         STA   $04
LP7      LDY   #$F0
SOUND    JSR   BSCSPK
         INY
         CPY   #$00
         BMI   SOUND
```

Il le sera autant de fois qu'il est nécessaire pour marquer le délai secondaire, spécifié par le registre Y. La constante de la fréquence est alors incrémentée, rangée à nouveau à l'adresse 04 et comparée à la fréquence maximum. Tant que cette dernière n'est pas atteinte, le programme continue à produire un son de fréquence croissante.

(1) Sur une carte d'expérimentation, on procéderait en recouvrant le détecteur avec le doigt, ou avec un morceau de tissu.

```
LDA   #$01
CLC
ADC   $04
STA   $04
CMP   #$A8
BMI   LP7
JMP   DETECT
```

Lorsqu'elle est enfin atteinte, le programme retourne à son point de départ. Plusieurs améliorations sont envisageables.

Dans le cadre d'une utilisation réaliste, le système d'alarme sera placé dans un endroit quelconque de la maison, de même que la paire photo-électrique, comportant l'émetteur de lumière et le détecteur. (En pratique, on utilise souvent un faisceau infrarouge, invisible à l'œil.) Le système peut, indifféremment, être chargé de protéger une pièce ou l'entrée de la maison. Il devra être modifié de façon que, lorsque l'alarme vient d'être mise en route, l'usager puisse quitter la maison sans la déclencher. Le premier exercice apportera cette amélioration :

Exercice 5-19 : *Modifiez le programme de telle sorte que l'utilisateur puisse sortir de la maison dans les deux minutes qui suivent l'armement (la mise en route) du système. En d'autres termes, aucune alarme ne devra être déclenchée dans les deux minutes qui suivent la mise en route, quel que soit l'état du détecteur. Après quoi, l'alarme devra fonctionner normalement.*

Autre problème : en rentrant chez lui, l'usager ne doit pas déclencher l'alarme immédiatement. Il faut lui donner le temps d'aller éteindre le micro-ordinateur. L'exercice suivant s'en charge :

Exercice 5-20 : *Une fois armée (après deux minutes), l'alarme ne doit retentir que 30 secondes après détection d'une entrée.*

Perfectionnons encore le système. Des variations mineures du faisceau lumineux peuvent provoquer du bruit sur la ligne, et déclencher l'alarme. Ce que nous voulons éviter.

Exercice 5-21 : *Modifiez le programme de sorte que l'alarme ne soit déclenchée que si l'interruption du faisceau dure plus de 0,05 seconde.*

Nouvelle amélioration : dans le cas où un animal déclencherait l'alarme, un système d'arrêt automatique doit être mis en place. L'alarme devra retentir deux minutes après détection de l'intrus, puis s'arrêter automatiquement.

Exercice 5-22 : *Modifiez le programme de façon que l'alarme sonne deux minutes après son déclenchement, puis s'arrête d'elle-même.*

Au moment de la détection d'un intrus, une action supplémentaire peut être envisagée. Par exemple : allumer les lumières ou téléphoner à la police. Cette opération peut être réalisée facilement. Il suffit d'activer un relais extérieur.

Exercice 5-23 : *Modifiez le programme ci-dessus de façon à actionner un relais extérieur, chaque fois qu'une entrée est détectée.*

Remarque : Cette possibilité peut être utilisée avantageusement, même si vous ne pouvez pas appeler la police automatiquement (les règlements français l'interdisent aux simples particuliers). Vous pourrez ainsi connecter une lampe au relais : elle vous indiquera, à votre retour, si un intrus est venu et s'est enfui rapidement.

Exercice 5-24 : *Pourriez-vous supprimer l'instruction CPY #*$00 à l'adresse 0299 ?

Exercice 5-25 : *Ajoutez un bouton « panique », grâce auquel vous pourrez déclencher l'alarme à tout moment. Modifiez le son de l'alarme, afin que les voisins puissent distinguer un « appel panique » d'une « alarme simple ».*

COMMANDE D'UN MOTEUR A COURANT CONTINU

Le but de ce programme est de contrôler la vitesse d'un moteur ordinaire à courant continu. Nous connecterons au micro-ordinateur un moteur 12 V courant continu, banal et peu coûteux. La vitesse de rotation sera spécifiée par des interrupteurs. Nous utiliserons trois interrupteurs, de façon à spécifier 8 combinaisons, correspondant à 8 vitesses de rotation. La figure 5-36 expose le circuit du moteur, et la figure 5-27 le branchement des interrupteurs.

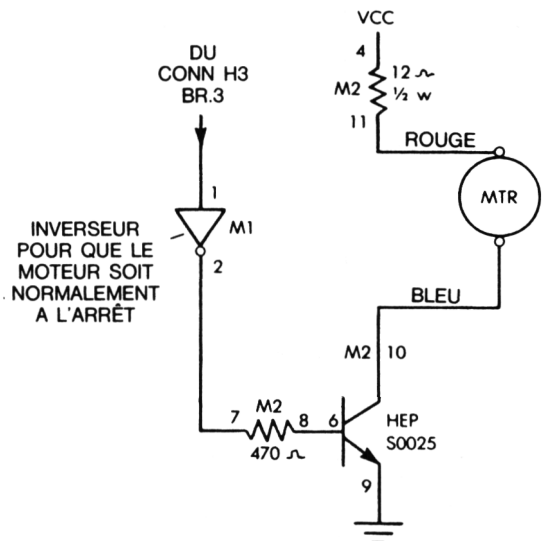


Fig. 5-36 : Circuit du moteur

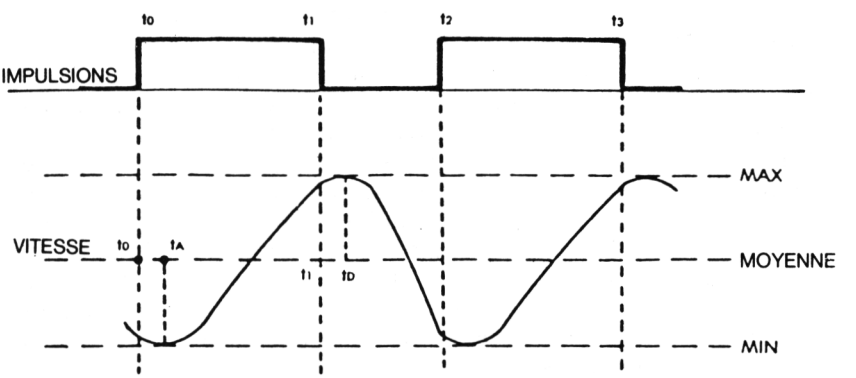


Fig. 5-37 : Contrôle digital de la vitesse

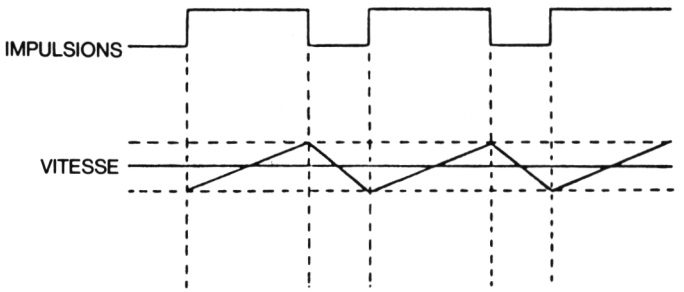


Fig. 5-38 : Diagramme de vitesse simplifié

Le principe utilisé pour contrôler la vitesse du moteur consiste à mettre ce dernier sous tension, pendant un temps déterminé, puis à l'arrêter. En raison de son inertie, le moteur continuera à tourner quelque temps. On enverra alors une nouvelle impulsion qui, de nouveau, mettra le moteur sous tension. Nouvelle accélération, et on recommence. La vitesse du moteur qui en résulte est montrée figure 5-37. La même courbe apparaît sous forme simplifiée figure 5-38. Il s'agit, essentiellement, d'une courbe en dents de scie, dans laquelle le moteur accélère tant qu'il est sous tension, puis ralentit jusqu'à ce qu'il reçoive l'impulsion suivante. La vitesse moyenne est marquée par la ligne horizontale entre les vitesse maximum et minimum de la figure 5-37. L'illustration montre que la vitesse oscillera, constamment, entre le maximum et le minimum. Pour que la vitesse soit définie avec précision, il faut que le maximum et le minimum soient rapprochés, ce qui est possible, en utilisant des impulsions courtes. Cependant, dans tout phénomène mettant en jeu de l'inertie et des oscillations, des instabilités peuvent se présenter. On notera, en particulier, sur l'illustration 5-37, que si la nouvelle impulsion est donnée avant l'instant T_D, alors la vitesse ne diminuera pas, mais, au contraire, augmentera. L'explication est simple : en raison de son inertie, le moteur n'a pas encore eu le temps de se ralentir. Des phénomènes encore plus complexes peuvent, le cas échéant, se produire. Nous ne les aborderons pas en détail ici. Nous nous bornerons à développer un programme comportant des délais que nous ajusterons ensuite, à tâtons, pour les faire convenir au type de moteur utilisé. Nous avertissons le lecteur que ces délais peuvent être ajustés de différentes manières, pour améliorer la précision de la vitesse et éliminer les problèmes d'oscillations.

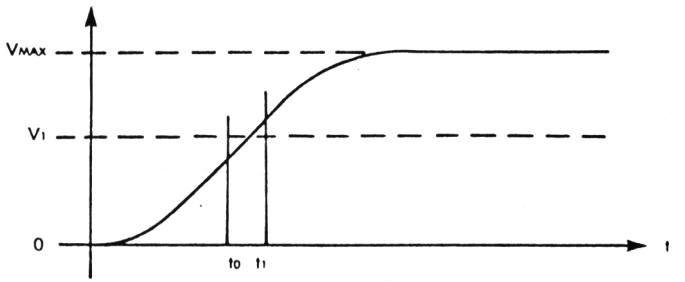


Fig. 5-39 : Courbe de vitesse d'un moteur à courant continu

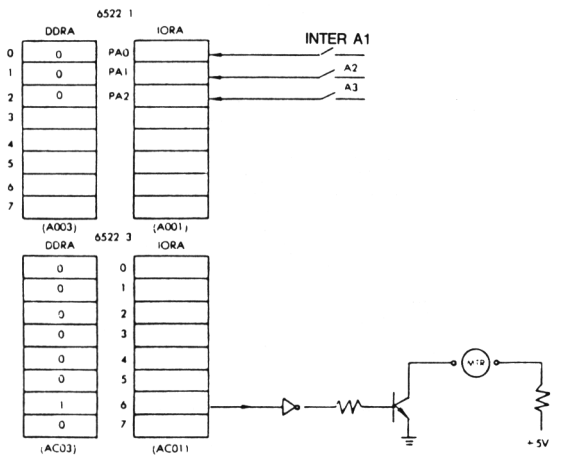


Fig. 5-40 : Les branchements

Les branchements hardware

Nous utiliserons deux ports : sur le 6522 n° 1, et sur le 6522 n° 2. (Cf. fig. 5-40). Le registre IORA du 6522 n° 1 sera utilisé comme port d'entrée pour les trois interrupteurs. L'état de ces derniers va déterminer la vitesse du moteur. La valeur correspondante de DDRA apparaît sur la partie gauche de l'illustration. Le registre IORA du 6522 n° 3 sert de sortie, pour contrôler la vitesse du moteur proprement dit. Le moteur est relié au bit 6 de IORA. Le détail de l'interface apparaît figure 5-36. L'amplificateur est nécessaire pour inverser le signal. Le transistor servira à fournir la quantité de courant requise.

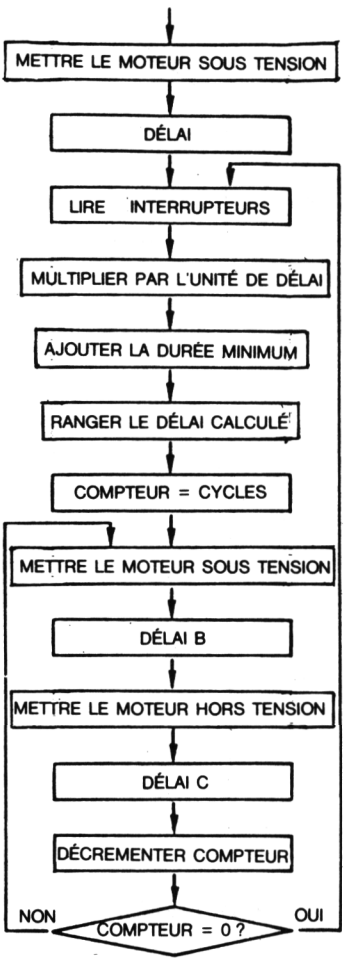


Fig. 5-41 : Ordinogramme du programme de moteur à courant continu

Le programme

La figure 5-41 montre l'ordinogramme du programme. Le moteur est mis sous tension pendant le temps T_{on}, et arrêté pendant le temps T_{off}. Dans cet algorithme, T_{off}, est fixe, tandis que T_{on} croît en fonction de la valeur des interrupteurs : de "000" à "111". La durée minimum de T_{on} correspond à l'état 000 des interrupteurs. Le

délai correspondant à une valeur des interrupteurs peut être calculé par la formule :

$$T_{on} = MIN + U \times inter$$

Numériquement, on utilisera les constantes :

$$T_{off} = C0_{HEXA} = 192 \text{ décimal}$$
$$T_{on} = 80_{HEXA} + inter \times 0B_{HEXA}$$
$$= 128 + inter \times 11 \text{ (décimal)}$$

INTERRUPTEURS	000	001	010	011	100	101	110	111
DÉLAI SOUS TENSION	128	139	150	161	172	183	197	205

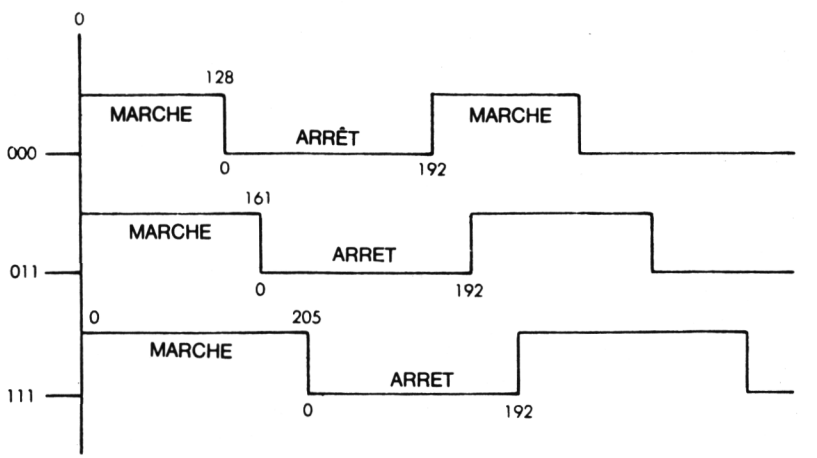


Fig. 5-42 : Les formes d'ondes

Les formes d'ondes correspondant aux différents états des interrupteurs apparaissent figure 5-42. Reportons-nous maintenant à l'ordinogramme de la figure 5-41. On commencera par mettre le moteur en marche, pendant un certain temps, pour obtenir une vitesse initiale de rotation (faute de quoi, une suite d'impulsions courtes ne seraient pas en mesure de lancer le moteur). On lira, ensuite, la valeur des interrupteurs qui, multipliée par l'unité de temps, sera ajoutée au minimum de durée d'impulsion. Le délai

ainsi calculé est mémorisé, et le moteur est mis en marche pendant un laps de temps correspondant à la durée calculée : c'est le délai B. Puis, le moteur est mis hors tension pendant le temps C. Ce processus est répété pendant plusieurs cycles pour permettre à la vitesse de se stabiliser. On relit alors les interrupteurs. Si leur état a changé, on génère la nouvelle vitesse. Notez que le délai créé par la répétition du cycle, avant la relecture, résout le problème du rebond des interrupteurs. Si aucun délai n'était mis en œuvre pour stabiliser la vitesse, il faudrait traiter le rebond des interrupteurs par hardware ou par software (cf. réf. C5 pour des détails sur le rebond).

Branchements : Connecteur A sur connecteur H2
Connecteur AA sur connecteur H3

Ce programme lit les interrupteurs A1-A3 pour définir la vitesse du moteur désirée et il fait tourner le moteur à cette vitesse.

Il utilise deux sous-programmes : DLYA et DLYB

```
02B0  A9 40      MOTOR LDA  #$40     Met DDRA du VIA n° 3 à $40 pour
02B2  8D 03 AC         STA  $AC03    avoir le bit de commande du moteur
                                     en sortie
02B5  A9 00            LDA  #$00     Met le moteur sous tension pendant le
                                     délai DLYA pour obtenir une vitesse
02B7  8D 01 AC         STA  $AC01    initiale
02BA  A9 FF            LDA  #$FF
02BC  85 00            STA  $00
02BE  20 20 01         JSR  DLYA
02C1  A9 00            LDA  #$00     Met DDRA du VIA n° 1 à 0 pour mode
                                     entrée
02C3  8D 03 A0         STA  $A003
02C6  AD 01 A0 MTRSP   LDA  $A001    Lit les interrupteurs
02C9  29 07            AND  #$07     Ignore les 5 bits les plus à gauche
02CB  A8               TAY           (Y) = état des interrupteurs
02CC  A9 0B            LDA  #$0B     Met la différence de délai sous-tension
                                     entre deux états consécutifs des inter. à
02CE  85 06            STA  $06      0B
02D0  C0 00      LP8   CPY  #$00
92D2  F0 07            BEQ  ONDLY
02D4  18               CLC
02D5  65 06            ADC  $06
02D7  88               DEY           Boucle jusqu'à ce que ($0006) = état des
                                     inter. × $0B
02D8  4C D0 02         JMP  LP8
02DB  85 06      ONDLY STA  $06
02DD  A9 80            LDA  #$80     Calcule la constante de délai sous-
                                     tension = 80 + (état des inter. × 0B)
```

Fig. 5-43 : Commande d'un moteur (Programme 5-8)

```
02DF 18                    CLC
02E0 65 06                 ADC  $06
02E2 85 06                 STA  $06      Range cette constante à l'adresse 0006
02E4 A0 C0                 LDY  #$C0
02E6 A5 06          MTRON  LDA  $06      Transfert de (0006) en 0004 avant appel
                                           de DLYB
02E8 85 04                 STA  $04
02EA A9 00                 LDA  #$00     Met le moteur sous tension
02EC 8D 01 AC              STA  $AC01
02EF 20 48 02              JSR  DLYB     Puis appelle DLYB
02F2 A9 C0                 LDA  #$C0     Met la constante de délai hors tension à
                                           C0 indépendamment de l'état des
                                           interrupteurs
02F4 85 04                 STA  $04      Range cette constante en 0004
02F6 A9 40          MTROFF LDA  #$40     Met le moteur hors tension
02F8 8D 01 AC              STA  $AC01
02FB 20 48 02              JSR  DLYB     Puis appelle DLYB
02FE 88                    DEY
02FF C0 00                 CPY  #$00     Répète cette séquence marche-arrêt
                                           jusqu'à ce que (Y) = 00
0301 30 E3                 BMI  MTRON
0303 4C C6 02              JMP  MTRSP    Lit les interrupteurs et recommence tout
```

Fig. 5-43 : (suite)

Le programme apparaît figure 5-43. Les quatre premières instructions mettent le moteur sous-tension, en positionnant le registre direction, et en envoyant "0" dans le registre d'E/S.

```
MOTOR    LDA   #$40
         STA   $AC03
         LDA   #$00
AC       STA   $AC01
```

On dépose une constante de temps égale à "FF" dans la case mémoire "00", d'après la convention de passage de paramètre établie par le sous-programme DLYA (cf. programme 5-1). Appelé, ce dernier réalise le délai initial, requis par le moteur pour atteindre sa vitesse initiale.

```
LDA   #$FF
STA   $00
JSR   DLYA
```

On lit l'état des interrupteurs :

```
        LDA   #$00
        STA   $A003
MTRSP   LDA   $A001
```

Et on extrait la valeur des trois derniers bits :

```
        AND   #$07        MASQUE
        TAY
```

Pour tous les états des interrupteurs, sauf "000", on ajoutera une durée à la valeur minimale "80", en hexadécimal. On sauve ainsi la valeur des interrupteurs dans le registre index Y, et on charge l'unité de délai dans la case-mémoire "06".

```
        LDA   #$0B
        STA   $06
```

LP8 est une boucle d'addition, qui ajoutera l'unité de délai autant de fois que le spécifie l'état des interrupteurs.

```
LP8     CPY   #$00
        BEQ   ONDLY
        CLC
        ADC   $06
        DEY
        JMP   LP8
```

Exercice 5-26 : *Peut-on modifier le code ci-dessus de façon à rendre inutile le CPY #$00 ? Pourquoi ?*

Nous arrivons, à présent, à ONDLY. La case-mémoire 06 contient désormais le délai additionnel de l'impulsion spécifiée par les interrupteurs. Nous l'ajouterons à la valeur minimale "80" hexadécimal :

```
ONDLY   STA   $06
        LDA   #$80
        CLC
        ADC   $06
        STA   $06
```

On charge le registre Y avec la valeur "C0" hexadécimal, qui indique le nombre de fois qu'il conviendra de mettre le moteur sous, puis hors tension :

```
        LDY   #$C0
```

Lorsqu'on arrive à l'adresse MTRON, la case-mémoire 06 contient la constante appropriée au marquage du délai « sous-tension ». On la transfère alors à l'adresse-mémoire 04, pour pouvoir utiliser le sous-programme DLYB.

Le moteur est mis sous tension. Nous marquons le délai :

```
MTRON   LDA   $06
        STA   $04
        LDA   #$00        MET LE MOTEUR
                          SOUS TENSION
        STA   $AC01
        JSR   DLYB
```

A ce point, la création d'un délai « hors tension » est indispensable. On chargera la valeur hexadécimale "C0" à l'adresse-mémoire 04. De son côté le moteur sera mis explicitement, hors tension. Le délai est réalisé par le sous-programme DLYB :

```
        DLA   #$C0
        STA   $04
MTROFF  LDA   #$40        MOTEUR HORS TENSION
        STA   $AC01
        JSR   DLYB
```

Le moteur est maintenant hors tension : on décrémente le compteur de boucle Y. Le registre index Y est utilisé pour compter le nombre de cycles à accomplir. Il a été chargé avec la valeur initiale "C0" hexadécimal, et devra être décrémenté à chaque fois que le moteur sera mis hors tension. Si la valeur "0" est atteinte, le programme revient au début, et lit le prochain état des interrupteurs. Si elle ne l'est pas, le programme rebouclera sur MTRON, pour refaire un cycle sous tension/hors tension :

```
        DEY
        CPY   #$00
        BMI   MTRON
        JMP   MTRSP
```

Considérons maintenant une série d'améliorations à ce programme.

Exercice 5-26 : *D'abord le style. Vous examinerez la partie du programme située entre les adresses 2D0 et 2D8. Pouvez-vous suggérer une amélioration d'écriture ? (Indication : il est possible d'économiser une instruction.)*

Exercice 5-27 : *Même question entre 02FF et 0303.*

Exercice 5-28 : *Cet exercice n'aura de sens, que si vous procédez à une expérience sur un moteur réel. Vous augmenterez progressivement le délai « hors tension » en changeant la constante voulue dans le programme. Que se passera-t-il ?*

Exercice 5-29 : *Même question en réduisant le délai « hors-tension ». Quel sera le problème ?*

Exercice 5-30 : *Un autre algorithme utilisable serait d'envoyer un nombre variable d'impulsions « sous tension » de durée constante, c'est-à-dire d'ajuster le délai « hors tension », plutôt que le délai « sous tension ». Pouvez-vous modifier le programme en conséquence ?*

Note importante : Chaque moteur a des caractéristiques différentes. Les constantes de temps du programme seront donc, de préférence, déterminées par une procédure d'essais à tâtons. Nous vous engageons vivement à modifier les différentes constantes utilisées, telles que la durée « sous tension » minimum, la durée « hors tension » minimum et les incréments de temps, jusqu'à obtenir les valeurs donnant les meilleurs résultats. De plus, si vous avez l'intention de charger le moteur en le reliant à un appareil, vous ne manquerez pas d'introduire des paramètres de frottement et d'inertie supplémentaires. Les moteurs de maquettes bon marché peuvent d'autre part être mal lubrifiés. Leurs frottements peuvent augmenter fortement, après une période de quelques semaines ou de quelques mois, et dans ce cas ils auront besoin d'un temps de lancement et d'impulsions plus longs. Toutefois, si vous connaissez les particularités de votre moteur, vous devriez arriver à ajuster les paramètres en conséquence.

Exercice 5-31 : *Qu'arrivera-t-il si vous envoyez des impulsions de mise sous tension très courtes ?*

Le programme ci-dessus forme une boucle de régulation ouverte, dans laquelle la vitesse du moteur est commandée sans être mesurée. Suggérez quelques améliorations.

Exercice 5-32 : *Affichez la vitesse théorique du moteur. Elle devrait être identique à l'état des interrupteurs. En d'autres termes, vous devriez pouvoir afficher simplement un chiffre compris entre 0 et 7.*

Poursuivons : nous allons maintenant implanter une véritable boucle fermée. Cet exercice intéressera particulièrement tous ceux qui veulent comprendre par exemple la méthode de régulation d'un disque. Une manière simple et efficace de mesurer la vitesse du moteur, est d'attacher un disque de carton à l'arbre-moteur. On y perfore un trou, appelé trou d'index. Vous placerez le disque de manière qu'il dispose d'un émetteur de lumière, d'un côté, et d'un récepteur de l'autre. Lorsque le trou passera en face de la diode émettrice de lumière, cette dernière éclairera le récepteur (1). Chaque fois que le récepteur sera éclairé on détectera une impulsion. En comptant le nombre d'impulsions par seconde, on obtiendra la vitesse exacte du moteur, en tours/seconde. En utilisant cette information, il est possible d'ajuster la durée, ou la fréquence, des impulsions, pour réguler la vitesse avec une grande précision. La comparaison entre cette technique et celle des disques souples s'arrête là. Dans un disque souple, la vitesse doit être régulée avec grande précision. Elle doit l'être de surcroît durant une fraction de tour, et non en fonction de la moyenne calculée sur plusieurs tours. Par suite, on utilisera, sur un disque, des informations supplémentaires, enregistrées sur une piste : les impulsions qui en résultent seront utilisées, pour ajuster la vitesse, même pendant une fraction de tour. Dans le cas de notre moteur, il est important de mesurer la vitesse réelle, puisque tout frottement, toute charge, va modifier sa vitesse de rotation. Toutes les techniques hardware et software nécessaires pour réaliser ces opérations ont déjà été exposées.

Exercice 5-33 : *Ecrivez le programme qui le réalise.*

CONVERSION ANALOGIQUE-DIGITALE (CAPTEUR DE TEMPÉRATURE)

Nous utiliserons ici une thermistance pour mesurer des températures, bien que tout autre capteur de température puisse aussi bien faire l'affaire. Dans une thermistance, la résistance change avec la température. Nous nous servirons de ce phénomène pour détecter les changements de température dans l'environnement, et prendre une série de décisions qui en découlent. Le problème principal est de mesurer par un nombre binaire une grandeur analogique qui, par définition, varie de façon continue : ici, la résistance de la thermistance. On appelle cela le problème de la

(1) C'est exactement ce qui se passe dans une unité de disques souples pour détecter le trou d'index.

conversion analogique-digitale. Il existe aujourd'hui des composants capables d'effectuer cette conversion, pratiquement, en un seul boîtier. Nous ferons, pour notre part, appel à une solution moins coûteuse (et plus éducative), en utilisant un convertisseur digital-analogique, et quelques amplis-op. La conversion analogique-digitale sera accomplie par programme (1).

Notre technique est dite des *approximations successives*. On génère une valeur binaire initiale, et on la convertit en analogique. Cette approximation analogique est alors comparée [à l'aide d'un comparateur], à la valeur générée par la thermistance. Le résultat de la comparaison, 0 ou 1, selon qu'elle est plus petite ou plus grande, servira à construire l'approximation suivante.

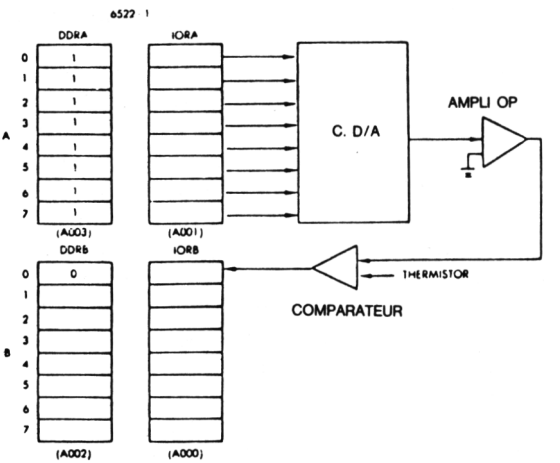


Fig. 5-44 : Connexion pour la conversion analogique-digitale

La figure 5-44 montre la connexion hardware utilisée dans cette expérience. Les 8 bits de sortie de IORA sont connectés à un convertisseur digital-analogique 8 bits. Ce dernier transformera le nombre binaire 8 bits en signal analogique, dont la valeur sera comparée à celle de la thermistance. La sortie du comparateur est connectée au bit 0 de IORB, où elle peut être testée.

L'algorithme met à 1, toute la série des bits de IORA, en commençant par le plus significatif (bit 7) jusqu'au bit 0.

La valeur initiale essayée sera "10000000". Si elle s'avère trop petite, on gardera le bit 7 inchangé en mettant à 1 le bit 6. Dans

(1) Pour plus de détails sur les techniques de conversion analogique-digitale, le lecteur se reportera au chapitre V de notre référence C5 : Techniques d'interface aux microprocesseurs.

cet exemple, l'approximation suivante sera "11000000". Si elle s'avère, cette fois, trop grande (on en décidera en examinant la sortie du comparateur). Il faudra mettre à 0 le bit 6. Nouvelle approximation : "10100000". Le bit 5 sera mis automatiquement à 1. Et ainsi de suite.

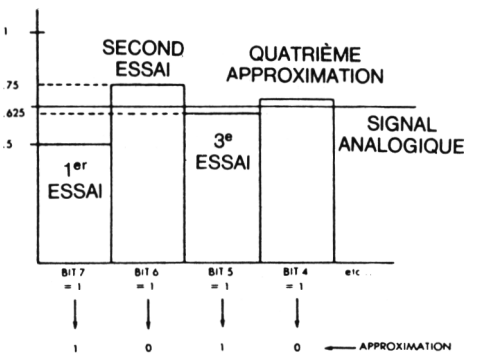


Fig. 5-45 : Approximations successives

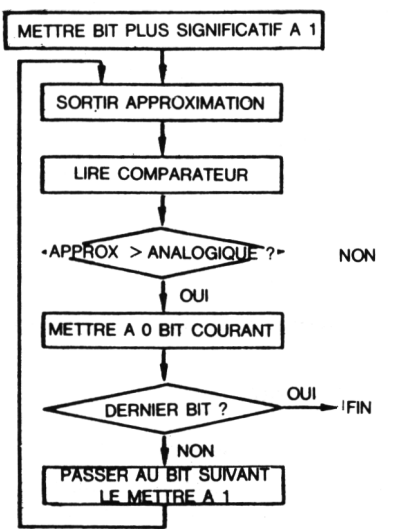


Fig. 5-46 : Ordinogramme des approximations successives

L'algorithme formel est illustré par la figure 5-45 et par l'ordinogramme de la figure 5-46. Le processus continuera, jusqu'à ce que les 8 bits aient tous été déterminés. La valeur qui en résulte est la meilleure approximation possible, permise par une représentation sur 8 bits. Naturellement, le procédé suppose que l'exécution de l'algorithme soit assez rapide, pour que la valeur analogique ne change pas plus vite qu'il n'est possible de la mesurer. Sinon, il faudrait utiliser un circuit échantillonneur bloqueur. La figure 5-45 montre les approximations successives se rapprochant de la valeur analogique exacte. Chaque fois qu'un nouveau bit est mis en jeu, l'intervalle est divisé par deux.

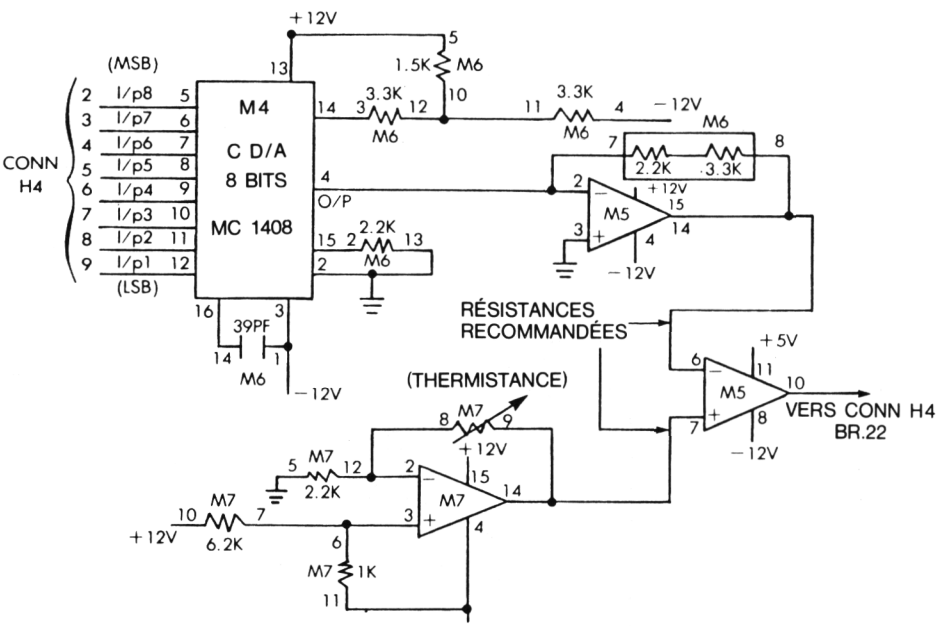


Fig. 5-47 : Interface de conversion analogique-digitale

Le branchement hardware

Les connexions hardware apparaissent sur les figures 5-47 et 5-48. Le convertisseur digital-analogique utilisé ici est un MC1408, qui nécessite une alimentation 12 V. La sortie commande l'ampli-op M5 alimentant l'entrée du comparateur. La sortie du comparateur est reliée à la broche 22 du connecteur H4. Elle va dans le bit 0 du registre IORB du 6522 n° 1.

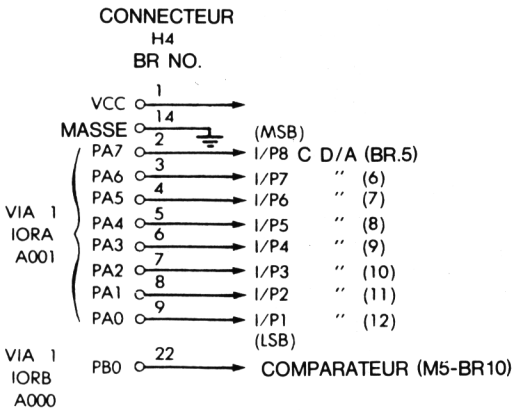


Fig. 5-48 : Liaison à H4

Le programme

La valeur de la température, mesurée sur la thermistance, sera indiquée par la fréquence d'un son produit par le haut-parleur. La hauteur du son sera d'autant plus grande que la température sera plus élevée.

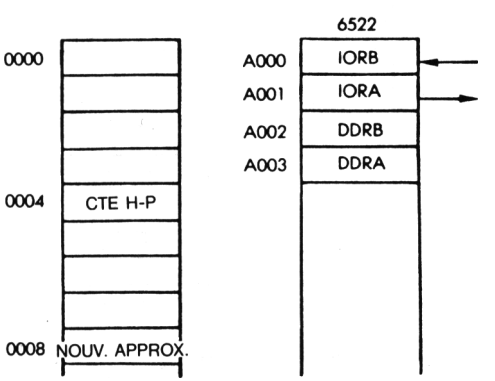


Fig. 5-49 : Carte d'implantation mémoire pour la conversion analogique-digitale

La figure 5-49 montre la carte d'implantation-mémoire du programme de conversion analogique-digitale. L'adresse-mémoire 4 sert à ranger la constante utilisée par le sous-programme DLYB, qui génère un délai spécifié par cette constante. L'adresse 8 sert

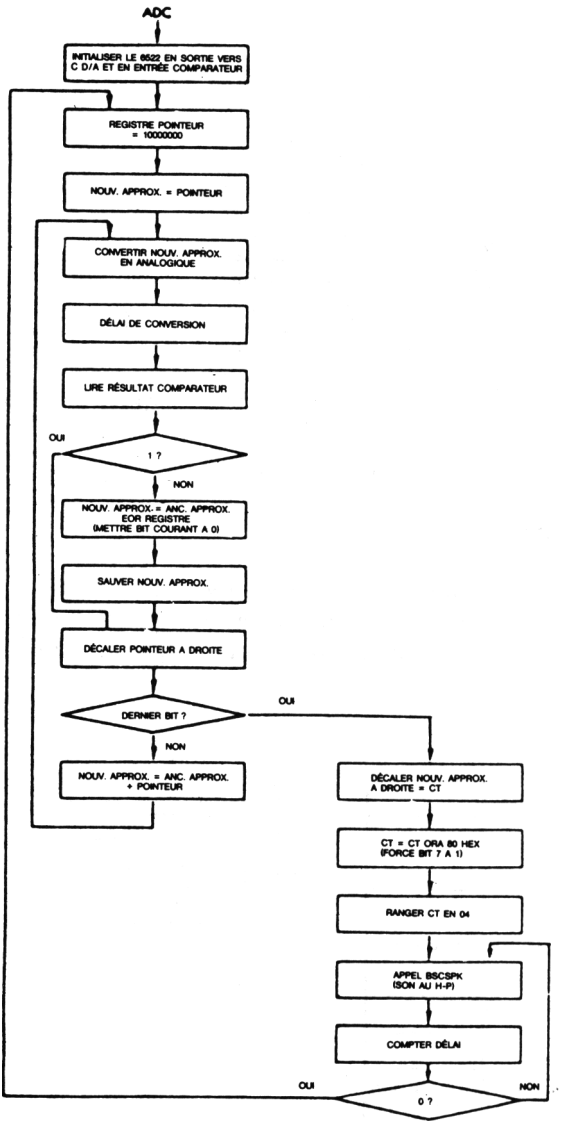


Fig. 5-50 : Ordinogramme de la conversion analogique-digitale

à ranger la nouvelle approximation calculée par le programme. Le 6522 n° 1 est aux adresses A000 et suivantes.

L'ordinogramme apparaît sur la figure 5-50. On commencera par initialiser le 6522, afin de configurer IORA en sortie, pour le convertisseur D/A. Le bit 0 de IORB est utilisé comme entrée du résultat du comparateur. Le registre pointeur est mis à la valeur initiale "10000000", qui est la valeur de la première approximation. Ce registre pointera vers le bit qui devra être mis à 1, dans la boucle des approximations successives. Le bit sera décalé à droite, à chaque itération.

La valeur initiale de l'approximation sera choisie égale au registre pointeur, avant d'être convertie en analogique. On marquera un délai pour donner le temps au convertisseur D/A d'effectuer sa conversion. Examinons la sortie du comparateur. Si elle est à 1, la nouvelle approximation est trop petite et, il est inutile de changer le bit courant. Si elle est à 0, l'approximation est trop grande, et il faut remettre à 0 le bit courant. Nous décalerons d'un cran à droite le pointeur, pour pointer vers le nouveau bit à mettre en jeu. L'approximation finale est calculée quand le dernier bit est atteint. S'il ne l'est pas on obtient l'ancienne approximation. Il faut alors démarrer une nouvelle itération.

A chaque approximation, un son sera produit dont la hauteur dépendra de la valeur de la mesure. On dispose d'une fréquence du son minimum, et la valeur de la fréquence à jouer sera obtenue en lui ajoutant la valeur de l'approximation. Le sous-programme BSCSPK sera ensuite appelé, pour faire retentir le haut-parleur. Après un certain temps d'émission du son, le programme lira, à nouveau, la valeur de la thermistance.

Sur la carte d'applications, la manière la plus rapide d'obtenir une réponse audible, est d'utiliser un fer à souder (ou une cigarette) et de l'approcher de la thermistance. La fréquence du son émis par le haut-parleur devrait monter rapidement, et redescendre lorsqu'on éloignera le fer à souder. Naturellement, la thermistance pourrait être localisée loin du micro-ordinateur. Convenablement isolée, elle pourrait être placée sur un mur, dans une tasse ou dans tout autre objet, dont on aurait l'intention de mesurer la température. Il serait également possible, pour connaître la température d'un liquide, d'y plonger un thermocouple, et d'autre part, de contrôler la température ambiante à l'aide par exemple d'un enroulement chauffant, branché sur l'un des relais. Resterait alors le problème de calibrer la thermistance de façon à réaliser des mesures précises.

Branchement : Connecteur A sur connecteur H4
Connecteur AA sur connecteur H3

Ce programme procède par approximations successives à l'aide d'un C.D/A afin de mesurer continûment la valeur analogique d'une thermistance. On utilise ensuite la valeur numérique approchée pour commander la fréquence du son d'un haut-parleur. D'après les variations de fréquence, on peut dire si la température monte ou descend.

La fréquence du haut-parleur est proportionnelle à la température (ou à la résistance de la thermistance).

Le programme utilise les sous-programmes BSCSPK et DLYB.

```
0360  A9 FF      ADC    LDA  #$FF     Met DDRA du VIA n° 1 à $FF pour être
0362  8D 03 A0          STA  $A003      en sortie afin de commander le C D/A

0365  A9 00             LDA  #$00     Met DDRB du VIA n° 1 à 0 pour être en
0367  8D 02 A0          STA  $A002      entrée afin de lire la sortie du
                                        comparateur
036A  A9 80      FSTBIT LDA  #$80     Met le bit le plus significatif à 1
036C  A8               TAY           (Y) contient le bit en cours de test
036D  85 08            STA  $08      La case mémoire 0008 contient la valeur
                                        courante
036F  A5 08      NXTBIT LDA  $08
0371  8D 01 A0          STA  $A001    Envoie la valeur courante sur le C. D/A
0374  A2 20             LDX  #$20     Délai de stabilisation du comparateur
0376  CA         LP9    DEX
0377  E0 00            CPX  #$00
0379  10 FB            BPL  LP9
037B  AD 00 A0          LDA  #$A000   Lit la sortie du comparateur
037E  29 01             AND  #$01     Prend le bit 0
0380  C9 01            CMP  #$01
0382  F0 05             BEQ  SHFBIT   Sortie comparateur = 1 signifie : valeur
                                        encore trop basse donc garder bit
                                        actuel et aller au suivant

0384  98               TYA           Sinon, valeur actuelle trop grande donc
0385  45 08            EOR  $08        enlever le bit courant
0387  85 08            STA  $08
0389  98        SHFBIT TYA           Décale (Y) à droite pour passer à
038A  4A               LSR  A          l'approximation suivante

038B  A8               TAY
038C  C9 00            CMP  #$00     (Y) = 0 signifie approximation terminée
038E  F0 08             BEQ  ECHO      donc aller activer le haut-parleur

0390  18               CLC
```

Fig. 5-51 : Conversion analogique-digitale (Programme 5-9)

```
0391  65  08            ADC  $08     Sortie vers le C.D/A en vue de la
                                      prochaine approximation = valeur
                                      courante plus bit suivant
0393  85  08            STA  $08
0395  4C  6F  03        JMP  NXTBIT
0398  A0  F0     ECHO   LDY  #$F0    Constante de durée pour chaque fré-
039A  A5  08            LDA  $08     quence
039C  4A                LSR  A
039D  85  04            STA  $04
039F  A9  80            LDA  #$80
03A1  05  04            ORA  $04     Calcule la fréquence correspondante

03A3  85  04            STA  $04     Et la range en 0004
03A5  20  30  02  SPKR  JSR  BSCSPK
03A8  88                DEY          Appelle BSCSPK pour activer le haut-
03A9  C0  00            CPY  #$00    parleur
03AB  30  F8            BMI  SPKR
03AD  4C  6A  03        JMP  FSTBIT  Recommence une nouvelle séquence
                                      d'approximation
```

Fig. 5-51 : (suite)

Examinons maintenant le programme (cf. fig. 5-51), et suggérons des améliorations. Les quatre premières instructions conditionnent les registres direction du port A et du port B du 6522 n° 1. Respectivement, en sortie (pour le convertisseur D/A), et en entrée (pour le comparateur) :

```
ADC     LDA   #$FF
        STA   $A003     DDRA 1 = FF = SORTIE
        LDA   #$00
        STA   $A002     DDRB 1 = 00 = ENTRÉE
```

Les deux instructions suivantes rangent la valeur hexadécimale "80" dans le registre Y. Le registre pointeur, reçoit la valeur initiale "10000000", en binaire.

```
FSTBIT  LDA   #$80
        TAY
```

La case-mémoire 08 a été réservée au stockage de l'approximation courante. Elle est initialisée à 10000000 :

```
        STA   $08
```

On entre, alors, dans la boucle d'itération principale. L'approximation binaire est extraite de la case-mémoire 08, et envoyée au convertisseur digital-analogique :

```
NXTBIT   LDA   $08
         STA   $A001
```

Un délai sera marqué pour permettre au comparateur de se stabiliser :

```
         LDX   #$20
LP9      DEX
         CPX   #$00
         BPL   LP9
```

La sortie du comparateur est lue :

```
         LDA   $A000        SORTIE
                            DU COMPARATEUR
```

On extrait et on teste, alors, le bit 0 :

```
         AND   #$01         BIT 0
         CMP   #$01
         BEQ   SHFBIT
```

Si la sortie est à 1, l'approximation sera encore trop petite : il faudra simplement mettre le bit suivant à 1. Si la sortie est à 0, la valeur sera trop grande, et on mettra à 0 le bit courant :

```
         TYA
         EOR   $08
         STA   $08
```

Après avoir ajusté, si nécessaire, l'approximation courante, on décalera, vers la droite, le registre pointeur, pour passer au bit suivant de l'approximation.

```
SHFBIT   TYA
         LSR A
```

La meilleure approximation possible est obtenue si l'on parvient à atteindre le dernier bit. On se branche, à ce moment, en ECHO, pour actionner le haut-parleur :

```
         TAY
         CMP   #$00
         BEQ   ECHO
```

Dans le cas contraire, on met à 1 le bit suivant de l'approximation, et on retourne au début de la boucle :

```
        CLC
        ADC   $08
        STA   $08
        JMP   NXTBIT
```

La routine ECHO actionne le haut-parleur en fonction de la valeur mesurée. Le registre Y est ici utilisé pour réaliser le délai de sonnerie du haut-parleur. Ce registre sera chargé avec la valeur initiale "F0" en hexadécimal. La valeur de l'approximation est lue dans la case-mémoire 08, décalée sur la droite de 1 bit. Cela veut dire que, dans cette technique, la valeur du dernier bit de l'approximation ne sera pas reflétée par une variation de la hauteur de la note.

On forcera le bit 7 à 1, pour s'assurer que le haut-parleur oscillera à une fréquence minimum audible.

La valeur résultante est rangée à l'adresse 04, qui sert à passer un paramètre à la routine BSCSPK qui a déjà été présentée :

```
ECHO    LDY   #$F0
        LDA   $08
        LSR   A
        STA   $04
        LDA   #$80
        ORA   $04
        STA   $04
SPKR    JSR   BSCSPK     ACTIONNER
                         LE HAUT-PARLEUR
```

Ensuite, on rappellera la routine qui actionne le haut-parleur à la fréquence spécifiée. On décrémentera et on testera le registre Y. Le haut-parleur retentira, tant que Y n'aura pas atteint la valeur 0.

```
        DEY
        CPY   #$00
        BMI   SPKR
        JMP   FSTBIT
```

Le haut-parleur a maintenant fini de retentir, et le programme va retourner au début de l'approximation, pour réexaminer l'état de la thermistance.

Exercice 5-34 : *Affichez, en hexadécimal, la valeur de l'approximation obtenue.*

Exercice 5-35 : *Est-il possible d'éliminer tous les CPY #$00 du programme ?*

Exercice 5-36 : *Calibrez votre thermistance, en déterminant quelle mesure calculée correspond à des températures données, mesurées avec un thermomètre. Rangez ces valeurs dans une table, afin d'afficher la température réelle, et non la valeur du registre d'approximation.*

Exercice 5-37 : *Modifiez le programme, de façon que le haut-parleur sonne de 1 à 10 fois. A haute température, il sonnera 10 fois. Il s'agit là d'un moyen de communiquer de façon audible les résultats de la mesure (avec une faible précision).*

Exercice 5-38 : *Ayant calibré votre thermistance, vous ajouterez un enroulement chauffant (qu'on peut se procurer à peu de frais, dans une quincaillerie), et régulerez la température d'un verre d'eau, de façon que l'eau reste précisément à la température T. Attention : la plupart des thermistances ne sont pas étanches. Il faut les attacher à l'extérieur du récipient, et non à l'intérieur. Vous pouvez aussi vous procurer des thermocouples, ou à défaut, certains types de thermistances, résistant à l'eau.*

Exercice 5-39 : *A titre d'amélioration supplémentaire apportée à votre routine d'alarme antivol (cf. programme 5-7), vous ajouterez à la boucle de contrôle de base, une routine qui vérifie périodiquement la température. Si cette dernière augmente au point de dépasser un niveau déterminé, disons 35 degrés centigrades, vous ferez sonner l'alarme. Ce dispositif est un détecteur d'incendie.*

Exercice 5-40 : *Autre variante : le but est, cette fois, de maintenir votre fer à souder à une certaine distance de la thermistance, pour l'amener à une température de, disons, 80° C. Vous modifierez votre programme de telle sorte qu'il fasse clignoter rapidement une LED, tant que la température de la thermistance reste très inférieure à la température désirée. La LED clignotera lentement à mesure qu'on s'approchera du niveau désiré. Une autre LED indiquera si l'on se trouve au-dessus ou en dessous de la température désirée.*

RÉCAPITULATION

Nous venons donc de développer une série d'applications du monde réel, allant du simple contrôle domestique à des commandes industrielles complexes. Nous avons connecté au micro-ordinateur une variété d'organes d'entrée-sortie : interrupteurs et LED, moteur à courant continu, thermistance, émetteur-récepteur de lumière, etc. Le choix des appareils et des techniques présentés devrait vous permettre de résoudre, à votre tour, un grand nombre de problèmes d'automatismes réels. Pour plus d'informations sur les techniques d'interface spécifiques, reportez-vous à notre référence C5 : *Techniques d'interface aux microprocesseurs*. Nous vous rappelons, en outre, que rien ne vaut l'expérience pratique pour véritablement développer vos capacités de programmation.

Au chapitre suivant, nous interfacerons de véritables périphériques d'ordinateur à la carte 6502.

6

PÉRIPHÉRIQUES

INTRODUCTION

Nous procéderons, ici, à la connexion de véritables périphériques d'ordinateur à la carte 6502. Les programmes de cette section ont été optimisés pour procéder à la démonstration de techniques « élégantes » de résolution des problèmes, faisant appel aux ressources spécifiques des composants mis en jeu. Nous connecterons, d'abord, un clavier matriciel 16 touches standard, et ferons un usage « astucieux » des propriétés du registre d'entrée-sortie, afin de limiter le nombre d'instructions nécessaires à l'identification et à l'affichage du caractère. Ensuite, nous fabriquerons un lecteur de ruban perforé artisanal, de coût minimal. Il suffira de tirer à la main le ruban perforé, en le faisant passer dans le lecteur, pour que les informations soient lues correctement par le micro-ordinateur. Enfin, nous mettrons en évidence la grande simplicité de l'opération qui consiste à connecter une micro-imprimante (ou un clavier ASCII) à un micro-ordinateur. A ce point, le lecteur devrait avoir pris confiance en lui : il a acquis, en principe, toutes les capacités qui lui permettront de résoudre la plupart des problèmes usuels rencontrés dans les applications réelles.

Les applications présentées ici sont utiles et faciles à réaliser. Les programmes sont directement applicables au SYM, au KIM ou à l'AIM65, avec des aménagements mineurs. Nous vous conseillerons donc, là encore, la pratique.

Tous les programmes sont courts. Ils vous apporteront des connaissances d'un intérêt indiscutable, même si vous n'avez pas l'intention de connecter un périphérique. La lecture attentive de ce chapitre est recommandée à tous.

CLAVIER

Nous allons, en premier lieu, connecter un clavier extérieur 16 touches en matrice (dit « clavier hexadécimal »), et identifier la touche sur laquelle on appuie. La figure 6-1 montre le branchement du clavier. Il est connecté aux 8 bits du registre IORA du 6522. Les bits 0 à 3 sont reliés aux lignes, et les bits 4 à 7 aux colonnes. Sur les schémas, on appuiera sur la touche à l'intersection de la ligne 2 et de la colonne 7. La ligne sera ainsi reliée à la colonne.

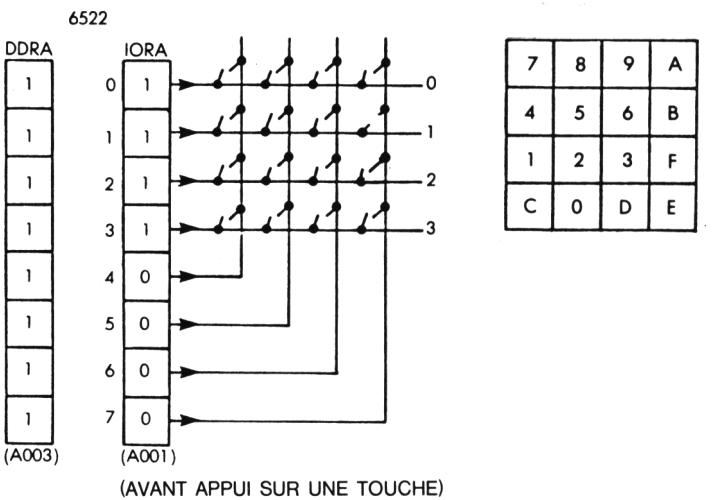


Fig. 6-1 : Branchement du clavier

Le registre direction est configuré avec tous les bits en sortie. Nous utiliserons la particularité suivante du 6522 : le registre IORA est, en fait, *bidirectionnel*. Toutes les lignes seront mises à 1, et toutes les colonnes à 0. Si on appuie sur une touche, la ligne correspondante va être mise à 0 par la colonne qui lui est reliée, au moyen de l'interrupteur. Lorsqu'on relira IORA, un 0 apparaîtra sur le bit correspondant à la ligne. Dans ce cas précis, en lisant IORA après appui sur la touche (2-7), on trouvera la valeur "00001011", en binaire, ou "0B", en hexadécimal. Faisant appel à une technique d' "inversion des lignes" (cf. nos références C4 et C5 pour des détails), nous écrirons : "11111011" en binaire, ou "FB" en hexadécimal, dans IORA. La ligne 2 est à 0, et va mettre également à 0 la colonne 7. En relisant IORA, on obtiendra la

valeur finale : "01111011" ("7B" en hexadécimal). A toute position de IORA où un bit est à 0, correspond une ligne, ou une colonne interconnectée. Cette technique détecte non seulement les touches sur lesquelles on a appuyé, mais aussi les erreurs. Elle signalera, par exemple, si on a appuyé de façon fautive sur plusieurs touches en même temps. Il y aura, dans ce cas, plusieurs 0 par groupe de 4 bits.

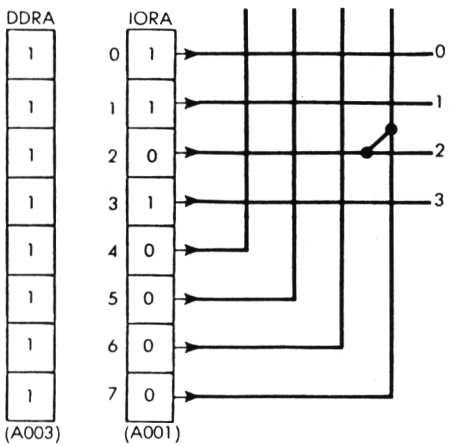


Fig. 6-2 : 2e étape : lecture de IORA après appui sur la touche

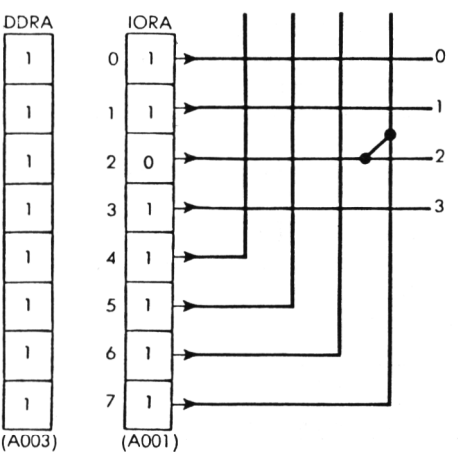


Fig. 6-3 : 3e étape : écriture de IORA

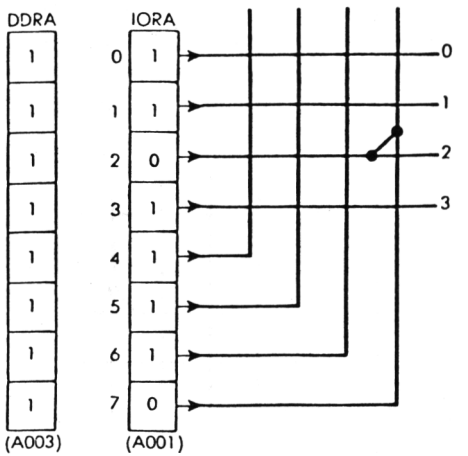


Fig. 6-4 : 4e étape : relecture de IORA

Pour identifier le caractère correspondant à la touche appuyée (caractère hexadécimal compris entre "0" et "F"), nous construirons, tout simplement, une table fournissant la représentation ASCII du caractère, pour chaque motif binaire correct trouvé dans IORA.

Exemple : nous venons de déterminer que lorsqu'on appuie sur "B", on obtient, dans IORA, le motif hexadécimal "7B". A titre d'exercice, le lecteur est invité à calculer le motif correspondant aux autres touches. La table de correspondance est donnée par la figure 6-5.

S'il se trouve, par hasard, un code illégal, on l'ignorera, et on réexplorera le clavier.

On peut afficher le code ASCII du caractère obtenu. Ici, nous utiliserons pour cela la routine d'affichage qui fait partie du moniteur du SYM. On suggérera, à la fin de cette section, d'afficher le caractère sur d'autres supports.

CARACTÈRE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
CODE	D7	EB	DB	BB	ED	DD	BD	EE	DE	BE	7E	7D	E7	B7	77	7B
ASCII	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46

Fig. 6-5 : Table des codes-caractères du clavier

Note : On utilisera, par commodité, les trois sous-programmes du moniteur : SCAND, HDOUT et ACCESS.

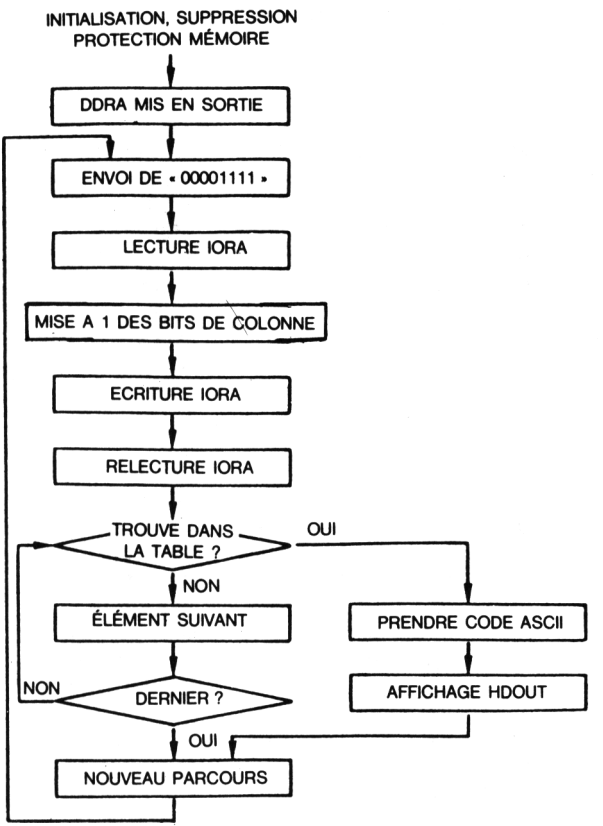


Fig. 6-6 : Ordinogramme du clavier

L'ordinogramme du programme apparaît figure 6-6.

Le programme commence par une initialisation. On envoie, ensuite, le motif hexadécimal « 0F » sur IORA. La valeur de IORA est relue (sans changer DDRA !). Il est inutile de ranger cette valeur dans un registre du 6502, ou en mémoire, en raison du caractère bidirectionnel de IORA sur le 6522. *Elle sera mémorisée dans le boîtier, et elle y restera.* Les quatre bits de colonne sont forcés à 1, et le nouveau motif renvoyé sur IORA. On relira IORA

pour obtenir le motif binaire final, et comparera le motif obtenu avec toutes les valeurs possibles de la table de la figure 6-5. S'il ne coïncide pas avec un élément de TAB, on comparera avec le suivant. Si aucun ne coïncide, on retournera au début de la boucle.

Le programme apparaît figure 6-7.

```
0000  20  86  8B  INIT    JSR   ACCESS
   3  A9  FF              LDA   #$FF
   5  8D  03  A0          STA   DDRA     DDRA = PAD
   8  A2  OF      START   LDX   #$0F
   A  8E  01  A0          STX   IORA     IORA = PA
   D  AD  01  A0          LDA   IORA     IORA = PA
0010  09  F0              ORA   #$F0
   2  8D  01  A0          STA   IORA     IORA = PA
   5  AD  01  A0          LDA   IORA     IORA = PA
   8  D5  30      LOOP    CMP   TAB, X
   A  F0  05              BEQ   DISPL
   C  CA                  DEX
   D  10  F9              BPL   LOOP
   F  30  05              BMI   SCAN
0021  B5  40      DISPL   LDA   ASCT, X
   3  20  00  89          JSR   HDOUT
   6  20  06  89  SCAN    JSR   SCAND
   9  4C  08  00          JMP   START
0030  EE  DE  BE  7E TAB  BYTE  $EE, $DE, $BE, $7E, $ED, $DD,
      ED  DD  BD  7D             $BD, $7D, $EB, $DB, $BB, $7B,
      EB  DB  BB  7B             $E7, $D7, $B7, $77
      E7  D7  B7  77
0040  37  38  39  41 ASCT BYTE  '7, '8, '9, 'A, '4, '5, '6,
      34  35  36  42             'B, '1, '2, '3, F, 'C, '0,
      31  32  33  46             'D, 'E
      43  30  44  45
```

Fig. 6-7 : Programme clavier (Programme 6-1)

Dans le cas du SYM, avant de mettre tous les bits du port A en sortie, la phase d'initialisation supprime la protection-mémoire, en faisant appel au sous-programme ACCESS :

```
IN      JSR   ACCESS
        LDA   #$FF        "11111111" = SORTIES
        STA   DDRA
```

On envoie le motif "00001111" au registre d'E/S :

```
START   LDX   #$0F        "00001111"
        STX   IORA
```

Il est immédiatement relu, et les bits de colonnes sont forcés à 1, en faisant le OU avec le motif "11110000" :

```
        LDA   IORA
        ORA   #$F0        "11110000"
```

Le motif résultant est envoyé sur le registre d'E/S (IORA) :

```
        STA   IORA
```

Il est immédiatement relu, et contient désormais le motif final, qui permettra de déterminer sur quelle touche on a appuyé :

```
        LDA   IORA
```

Le code contenu dans l'accumulateur va maintenant être comparé, en séquence, à chaque élément de la table. Lorsqu'on a affaire à une structure de table, il est commode d'utiliser le mode d'*adressage indexé,* pour accéder séquentiellement aux éléments. La valeur initiale du registre index est "0F" hexadécimal (15 décimal). On testera la coïncidence pour le dernier élément de la table (cf. fig. 6-7). puis le précédent. Si une coïncidence est trouvée, il faut se brancher à l'adresse DISPL :

```
LOOP    CMP   TAB, X
        BEQ   DISPL
        DEX
        BPL   LOOP
```

Faute de coïncidence, on décrémentera le registre index X, en vue de la prochaine comparaison. X doit être testé par rapport à "0". S'il décroît en dessous de 0, et devient négatif, c'est qu'aucune touche valide n'a été détectée. Il faut donc recommencer le balayage :

```
        BMI   SCAN
```

A ce point, le registre X indique le caractère reconnu. Il contient un nombre compris entre 0 et 15. Nous nous efforcerons main-

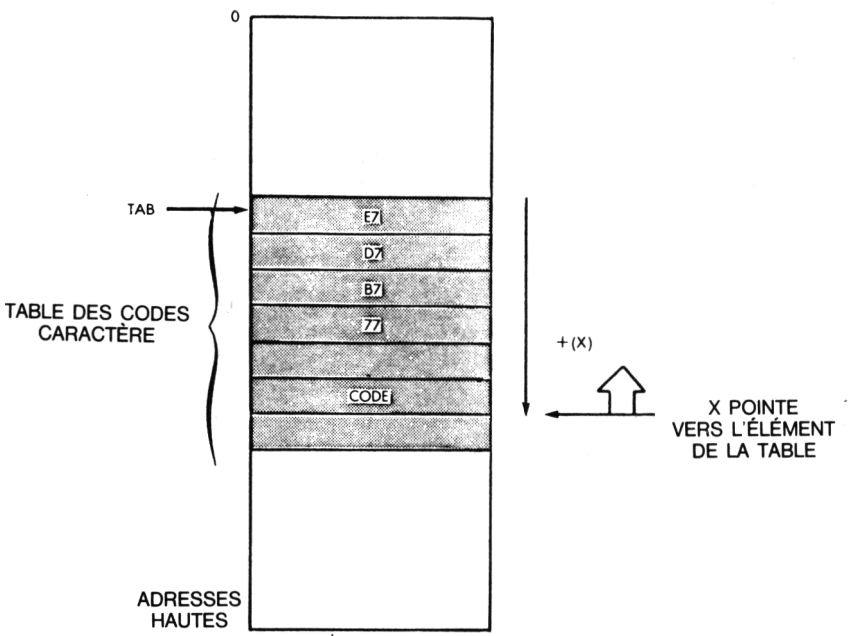


Fig. 6-8 : Accès à une table par adressage indexé

tenant de convertir ce nombre en code ASCII, condition nécessaire pour afficher (ou imprimer) le caractère reconnu :

```
DISPL  LDA     ASCT, X
```

A l'adresse DISPL, il convient de charger l'accumulateur avec la valeur du caractère déterminé par le contenu du registre index X. On aura, à nouveau, recours à l'*adressage indexé* pour ces données séquentielles (cf. fig. 6-9). On utilise alors le sous-programme HDOUT (du SYM) et on affiche le caractère (routine SCAND du SYM), avant de recommencer le balayage du clavier :

```
        JSR     HDOUT
SCAN    JSR     SCAND
        JMP     START
```

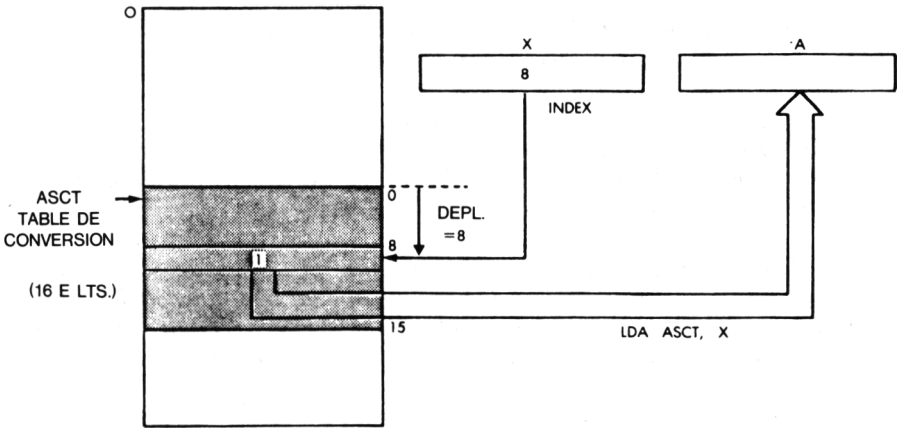


Fig. 6-3 : Conversion de l'identificateur-caractère en ASCII

Le programme se sert de deux tables de constantes. La première est appelée "TAB". Elle contient la liste des motifs binaires corrects de IORA. La valeur du registre index X, au moment de la lecture de l'un de ses éléments, détermine quelle touche a été enfoncée. La seconde table est appelée "ASCT". Elle contient le code ASCII de chacun des caractères du clavier.

Ces deux tables apparaissent à la fin du programme de la figure 6-7. Notez que le registre index X *n'a pas mission de contenir la valeur hexadécimale exacte,* correspondant à la touche enfoncée. Dès lors que les deux tables sont arrangées dans le même ordre, on extraira forcément le bon code ASCII, pour chaque motif légal trouvé dans TAB. C'est pourquoi les deux tables ne suivent pas l'ordre hexadécimal.

Exercice 6-1 : *Arrangez les deux tables TAB et ASCT de la figure 6-7, de telle sorte que la valeur du registre index soit toujours égale à la valeur hexadécimale de la touche qui a été enfoncée.*

Exercice 6-2 : *Au lieu de la méthode ci-dessus, arrangez les touches du clavier sans changer les tables TAB et ASCT, de sorte que la valeur du registre index X corresponde à la touche enfoncée.*

Nous suggérerons maintenant d'autres possibilités d'affichage à l'extérieur du caractère reconnu.

Exercice 6-3 : *Faites retentir le haut-parleur, une fois, si le caractère "1" a été tapé, deux fois si c'est le caractère "2", etc.*

Exercice 6-4 : *A l'aide du programme de Morse développé au chapitre 4 (cf. programme 4-3), modifiez le programme ci-dessus, de façon qu'il produise le code Morse correspondant à chaque touche enfoncée.*

Exercice 6-5 : *Modifiez le programme ci-dessus de façon qu'il produise une note, chaque fois qu'on appuie sur une touche. Une touche doit être chargée de produire les silences. D'autres peuvent être utilisées pour déterminer la durée de la note (durées 1, 2 et 4).*

Exercice 6-6 : *Ecrivez un programme de musique enregistrée. Vous commencerez par jouer un air, en tapant sur les touches du clavier, dans le bon ordre. Les 50 premières notes (ou tout autre nombre) devront être stockées, dans la mémoire du système. Puis vous taperez sur une touche spéciale, et le programme devra rejouer l'air mémorisé.*

LECTEUR DE RUBAN PERFORÉ OU CLAVIER ASCII

La connexion d'un clavier ASCII décodé, et celle d'un lecteur de ruban perforé, mettent en jeu des techniques presque identiques. L'interface hardware comporte 8 bits de données (le code ASCII 7 bits, plus la parité) et, en outre, un bit d'état indiquant qu'un caractère est prêt. Nous présenterons ici, une interface simple, à l'usage d'un lecteur de ruban artisanal simplifié. Le programme sera identique pour un clavier décodé.

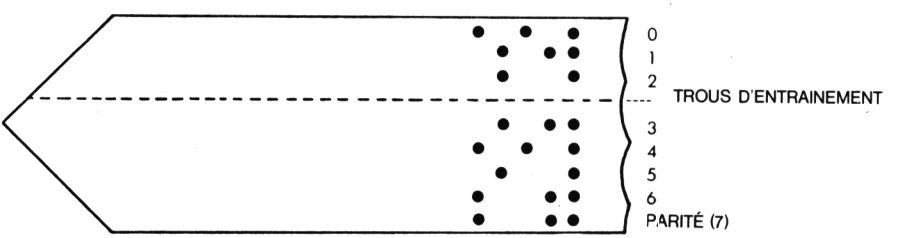
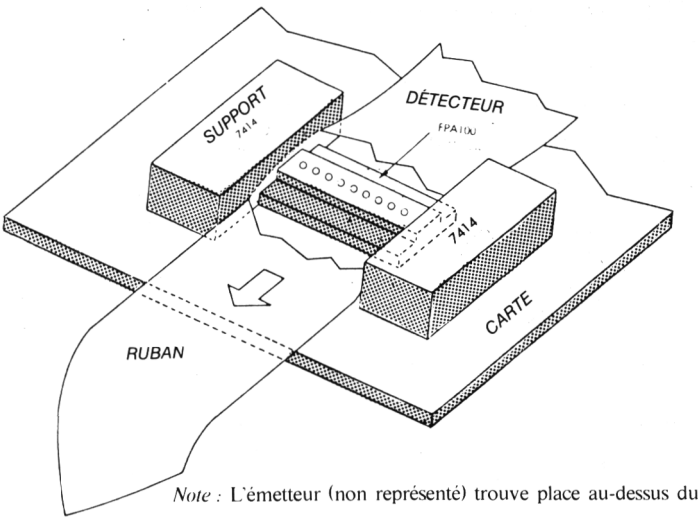


Fig. 6-10 : Ruban perforé 8 canaux

Le ruban perforé est traditionnellement utilisé pour stocker des programmes, sous une forme fiable et économique. Chaque caractère est représenté, sur le ruban de papier, par une rangée de trous (cf. fig. 6-10). Un trou, plus petit que les autres, sert à la roue



Note : L'émetteur (non représenté) trouve place au-dessus du détecteur.

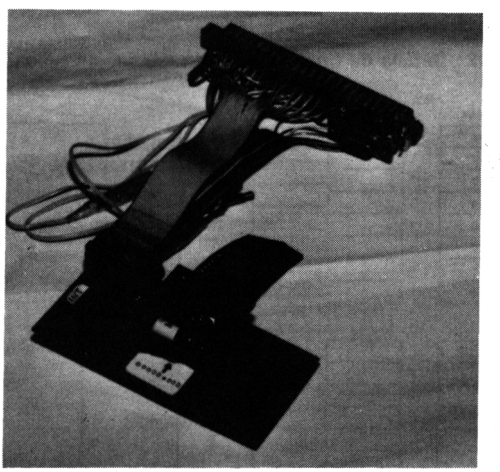


Fig. 6-11 : Hardware du lecteur de ruban perforé

L'émetteur FPA 100 est localisé sur le dessus de la petite carte. Le lecteur est connecté à la carte 6502 par un câble plat relié au connecteur A (en haut).

dentée, qui fait avancer le ruban. Les 8 autres trous (1) sont utilisés pour représenter, en code ASCII, le caractère proprement dit. Le ruban perforé est avancé d'une rangée à la fois. Le code correspondant à la rangée de trous sera lu par le lecteur. Nous utiliserons ici une paire d'émetteurs et de photodétecteurs FPA 100 (Fairchild).

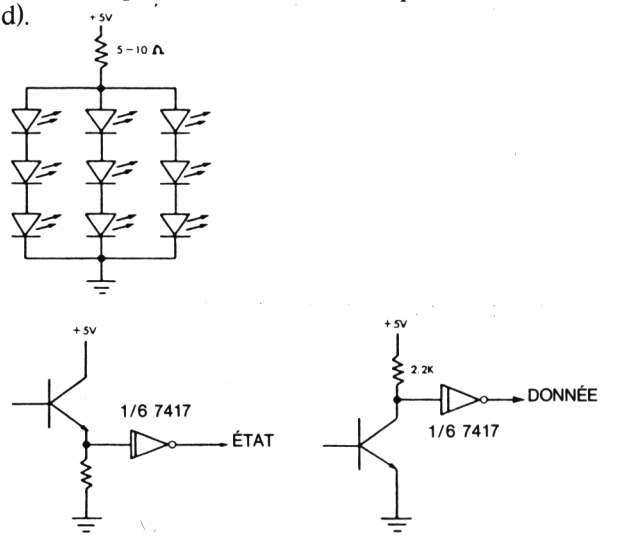


Fig. 6-12 : Détails de branchement du lecteur de ruban perforé

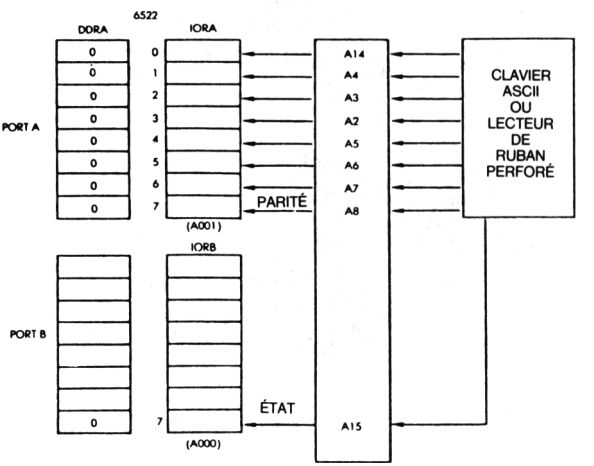


Fig. 6-13 : Interface du lecteur de ruban perforé

(1) Il existe d'autres codages utilisant moins de trous.

Les diodes émettent continuellement de la lumière. Lorsqu'un trou coïncide avec l'une d'elles, la lumière passe, et le photo-détecteur, placé de l'autre côté, la détecte. Cela fournit un "1". Quand la lumière est occultée, on reçoit un "0". Notez que l'intensité des LED doit être ajustée avec soin de façon que la lumière ne passe pas à travers le papier, en l'absence de trous (on présentera plus loin des remarques d'ordre pratique). Ce lecteur de ruban, très simple et très économique, fonctionne manuellement : on tire le ruban à la main, entre l'émetteur et le détecteur. Comme nous le verrons, le programme se synchronise par rapport aux trous prévus, normalement, pour la roue d'entraînement. Le schéma hardware apparaît à la figure 6-11, et les détails de branchement des diodes émettrices de lumière, des détecteurs de trous et des détecteurs de données à la figure 6-12. L'interface, avec le micro-ordinateur, est montrée figure 6-13. On utilisera, pour entrer, les bits de données du registre IORA du 6522 n° 1. Le registre IORB du même 6522 est utilisé pour lire le bit d'état sur sa position 7.

Les signaux sont mis en forme par des bascules de Schmidt (7414). Les deux supports des 7414 servent de guides au ruban perforé. Le signal correspondant à la détection d'un *trou*

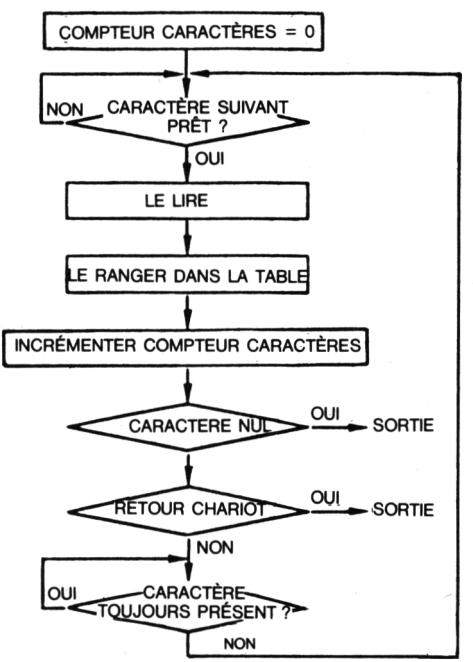


Fig. 6-14 : Ordinogramme lecture de ruban perforé

d'entraînement est "0". Le signal correspondant à un *trou de donnée* est "1".

A signaler que, dans cette interface simplifiée, on n'utilise qu'une seule résistance pour commander toutes les LED. Dans la pratique, on utiliserait probablement une résistance individuelle pour chaque LED. La valeur de la résistance doit être ajustée soigneusement, de sorte qu'il ne passe par les trous que l'intensité de lumière tout juste suffisante pour être décelée par les détecteurs. Autrement, dans la mesure où le ruban de papier usuel est assez transparent, on ne détecterait partout que, des 1 ("11111111"). La lumière passerait en effet même en l'absence de trou. Si vous rencontrez des problèmes avec la valeur de la résistance, essayez d'utiliser un ruban de papier noir, ou à la rigueur très opaque.

La figure 6-14 montre l'ordinogramme du programme. Nous utiliserons un compteur, pour dénombrer les caractères à leur arrivée. Le programme restera dans une boucle d'attente, jusqu'à ce que le caractère suivant soit prêt. Il le sera lorsqu'un trou d'entraînement se trouvera en face du détecteur correspondant. Le caractère devra être lu rapidement, dès que le signal d'état en aura révélé la présence. Il sera alors rangé dans une table, en mémoire, et on procédera à l'incrémentation du compteur de caractères.

Par convention l'opération de lecture se terminera, soit par un caractère "nul" (aucune perforation sur la bande), soit par un caractère de "retour chariot". S'il s'en trouve un, le programme s'arrêtera. Sinon il reviendra au début de la boucle, non sans avoir attendu que le bit d'état ait été restauré. Une fois le signal "caractère prêt" disparu, le programme reviendra au début et attendra que le caractère suivant soit prêt, à son tour.

La figure 6-15 expose l'implantation mémoire de ce programme. Le programme lui-même, apparaît figure 6-16.

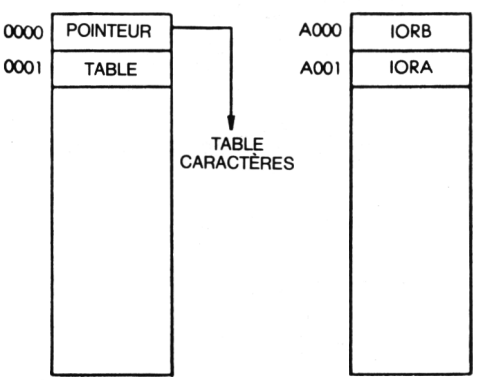


Fig. 6-15 : Carte d'implantation mémoire pour le lecteur de ruban perforé

```
0002  A0  00        KBPT  LDY  #0
   4  2C  00  A0    TS    BIT  IORB
   7  30  FB              BMI  TS
   9  AD  01  A0          LDA  IORA
   C  91  00              STA  ($00), Y
   E  C8                  INY
   F  C9  00              CMP  #0
0011  F0  0B              BEQ  RET
   3  C9  8D              CMP  #$8D
   5  FO  07              BEQ  RET
   7  2C  00  A0    TE    BIT  IORB
   A  10  FB              BPL  TE
   C  30  E6              BMI  TS
   E  60          RET     RTS
```

Fig. 6-16 : Programme pour lecteur de ruban perforé ou clavier décodé (Programme 6-2)

Le programme suppose que DDRA et DDRB ont été initialisés aux valeurs convenables. Dans le cas contraire, il faudrait ajouter des lignes d'initialisation au début du programme. On utilisera comme compteur de caractères le registre Y, initialisé à "0" :

```
KBPT    LDY   #0
```

On testera la valeur du bit d'état, pour déterminer si un caractère est prêt. Il sera connecté au bit 7 de IORB, pour faciliter son examen :

```
TS      BIT   IORB
        BMI   TS
```

Le bit 7 est en position privilégiée pour connecter un signal d'état, étant donné que cette position peut être testée en une seule instruction : le bit 7 est le bit de "signe". Il conditionne l'indicateur "N" du registre d'état, qui peut être testé directement pour déterminer s'il est "positif" ou négatif ("0" ou "1"). Nous utiliserons l'instruction BMI (branchement si négatif). Tant que le signal est à "1", aucun caractère n'est prêt. Quand il passe à "0", un caractère

est disponible. Il est alors possible de charger l'accumulateur avec les données présentes sur les lignes de données :

```
LDA   IORA          LIRE LES DONNÉES
```

Le caractère 8 bits obtenu du lecteur de ruban perforé devra être stocké dans un emplacement-mémoire convenable. Nous supposerons que l'adresse de départ du tampon de ligne a été déposée aux adresses 00 et 01, et utiliserons l'adressage indirect pour accéder à la table. En outre, l'adressage sera indexé par Y de façon à accéder, successivement, à tous les éléments de la table. L'instruction correspondante est :

```
STA   ($00), Y
```

Examinons cette instruction en mode indirect indexé. *L'indirection* signifie : "Allez à l'adresse mémoire 00, et utilisez son contenu comme une adresse." (Etape 1 de la figure 6-17.)

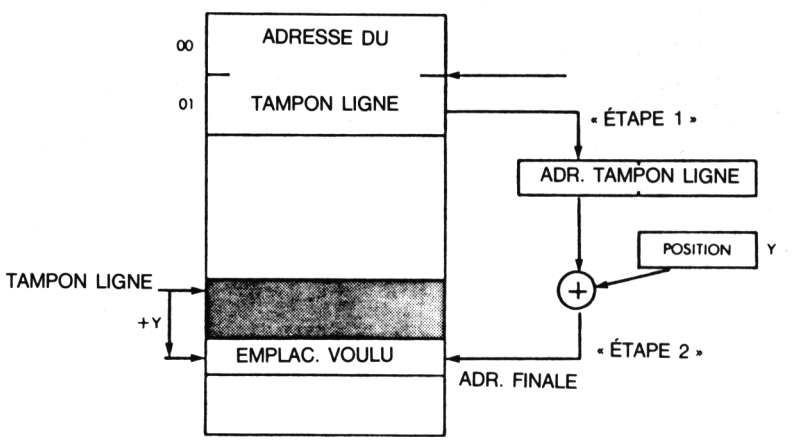


Fig. 6-17 : Accès indirect indexé : STA ($00), Y

Le registre Y servira d'index : son contenu sera ajouté à l'adresse de base, pour fournir l'adresse finale (étape 2 de la figure 6-17),

et jouera le rôle de déplacement dans la table-tampon. Il sera, en d'autres termes, le pointeur vers l'élément courant.

Le compteur caractères est incrémenté, pointant ainsi vers le prochain emplacement libre du tampon-ligne, en prévision du caractère suivant :

```
        INY
```

Testons, maintenant, le caractère dans l'accumulateur, pour déterminer s'il s'agit d'un "nul" ou d'un "retour chariot", ce qui marquerait la fin de la ligne. Nous nous servirons des quatre instructions suivantes :

```
        CMP   #0        NUL ?
        BEQ   RET       SI OUI, SORTIE
        CMP   #$8D      RETOUR CHARIOT ?
        BEQ   RET       SI OUI, RETOUR
```

Il est nécessaire d'attendre que le signal "caractère prêt" disparaisse, avant de le tester à nouveau, faute de quoi nous lirions deux fois le même caractère. (Cf. l'ordinogramme de la figure 6-14.) Nous utiliserons les trois instructions suivantes :

```
TE      BIT   IORB      TEST DU SIGNAL PRÊT
        BPL   TE
        BMI   TS
```

Finalement, le sous-programme se termine par l'instruction de retour habituelle :

```
RET     RTS
```

Exercice 6-7 : *En plus de ranger le caractère dans une table, générez, à l'aide du haut-parleur, le code Morse correspondant au caractère lu. Prenez soin de générer le Morse assez vite pour tenir le rythme de l'arrivée des caractères. Vous pouvez également choisir de tirer le ruban très lentement pour disposer de plus de temps entre deux caractères. Autre solution possible : vous pouvez générer le Morse à la fin d'une ligne seulement, quand tous les caractères ont été lus. C'est, nettement, la solution la plus sûre, mais elle vous prive du plaisir de vérifier, tout de suite, que chaque caractère est correctement lu !*

Exercice 6-8 : *Connectez 8 LED sur la carte du lecteur de ruban, et faites-les allumer par le 6502, à mesure que chaque caractère est reconnu.*

Exercice 6-9 : *Faites retentir une alarme si le bit de parité est incorrect. (Le bit de parité s'assure que le nombre total de bits mis à 1 pour un caractère donné, est pair ou impair, selon la convention adoptée. Vous devrez déterminer celle-ci.)*

MICRO-IMPRIMANTE

De nombreuses petites imprimantes utilisent un papier électrosensible, et composent 20 caractères, formés à l'aide d'une matrice de points, par ligne. Il existe, notamment, différents modèles Olivetti ou Matsushita (cf. fig. 3-19). L'imprimante simple nécessite une petite interface, qui envoie les signaux appropriés à la tête d'impression, déplace le papier et gère les ressources du mécanisme d'impression. Une fois munie d'une telle interface élémentaire, *la micro-imprimante peut être connectée à n'importe quel micro-ordinateur équipé d'un PIO* (port d'entrées-sorties programmable). Nous utiliserons ce type d'imprimante, connectée au système 6502 par un 6522, et un port de 6532. Des disparités peuvent se présenter, si vous utilisez une imprimante munie d'une interface différente. Mais la logique du programme devrait être analogue, pour l'essentiel.

Le programme imprime, en une seule fois, une ligne de 20 caractères. Il fournit le signal "départ impression", puis envoie les 20 caractères à la suite, en attendant à chaque fois que l'interface

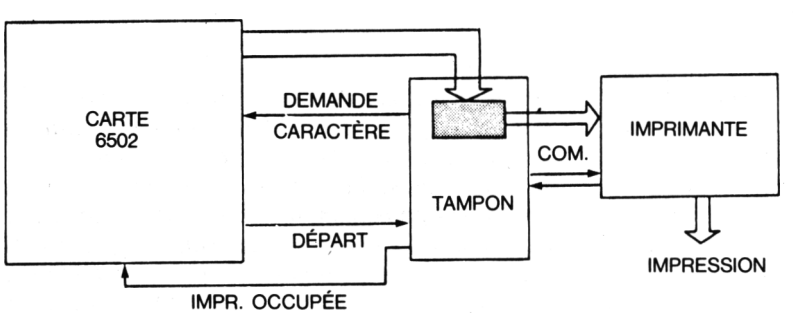


Fig. 6-18 : Interface élémentaire pour imprimante

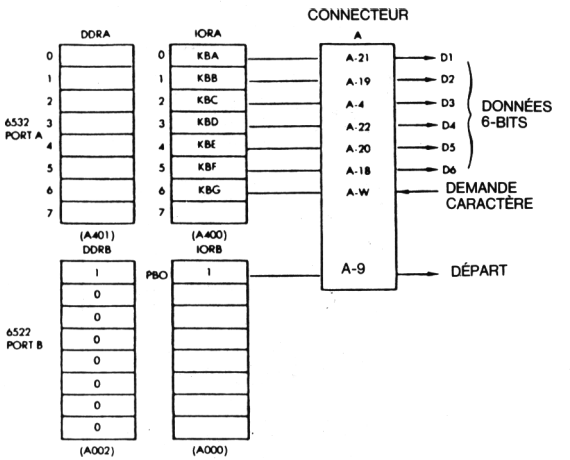


Fig. 6-19 : Branchement de l'imprimante

fournisse un signal "demande de caractère". En réponse à ce signal, le programme doit rapidement fournir le caractère voulu, sinon c'est le caractère précédent, mémorisé dans le tampon de l'interface, qui sera imprimé par erreur. Le caractère est fourni sur les 6 lignes de données : on utilise une représentation des caractères sur 6 bits (cf. fig. 6-18).

Le branchement hardware de l'imprimante apparaît figure 6-19. On utilisera le port A du 6532, et le bit 0 du port B du 6522. L'IORA du 6532 fournit les 6 lignes de données, et reçoit, sur le bit 6, le signal "demande caractère", comme le montre l'illustration. On utilisera le bit 0 de IORB du 6522 pour générer le signal de "départ". De surcroît, l'interface fournit, normalement, un signal "imprimante occupée", dont on ne tiendra pas compte ici, et qu'on remplacera par une routine de délai de 50 millisecondes. La figure 6-20 montre l'ordinogramme du programme.

Initialisons les registres direction des deux ports. Nous générerons une impulsion de départ, pour démarrer l'imprimante. Le programme testera la ligne "demande caractère", attendra, à cet endroit, qu'une transition indique une demande de caractère et extraira le caractère suivant de la zone-mémoire où est rangée la ligne de 20 caractères (cf. fig. 6-21). Quand le caractère sera envoyé à l'imprimante, le programme attendra que le signal "demande

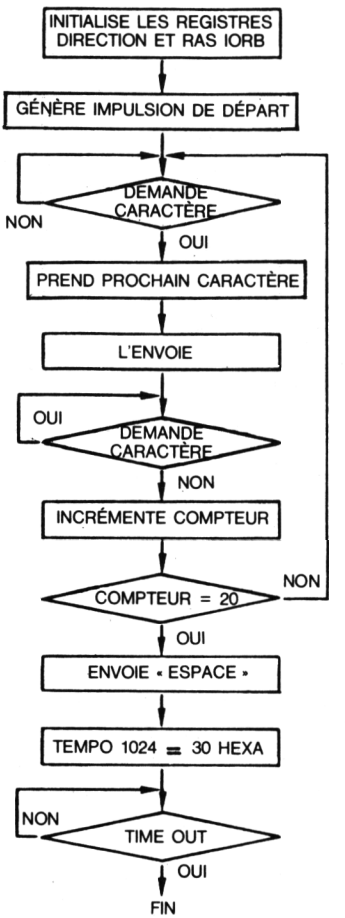


Fig. 6-20 : Ordinogramme par le programme d'imprimante

caractère" disparaisse, incrémentera le compteur caractères, et procédera à un test, pour vérifier qu'il a atteint la valeur 20. Si tel n'est pas le cas, un autre caractère sera envoyé à l'imprimante, et on retournera au début de la boucle. Une fois les 20 caractères envoyés, on expédie un code « espace » pour terminer la ligne. Il faut alors marquer un délai de 48 millisecondes, pour permettre aux éléments mécaniques de l'imprimante de se positionner, en vue de la ligne suivante. Nous utiliserons, pour cela, le temporisateur

du 6532, et ferons appel à l'adresse correspondant à un taux de division par 1 024. Le facteur 1 024 correspond, lui-même, à un délai de 1 024 microsecondes, soit, approximativement, 1 milliseconde par unité de comptage. Le compteur est chargé avec la valeur "30" hexa = 48 décimal.

On sortira du programme, à l'issue du time-out.

Le programme apparaît figure 6-22. La carte d'implantation-mémoire est fournie par la figure 6-21. Les deux cases-mémoire "00" et "01" contiennent le pointeur dirigé vers le premier caractère de la ligne. Pour utiliser ce programme, l'utilisateur devra écrire la valeur "01" à l'adresse A002 (DDRB), et "00" en A000 (IORB), avant de mettre l'imprimante sous tension. Les adresses-mémoire où sont implantés les composants d'entrées-sorties apparaissent sur la droite de la figure 6-19. Examinons le programme.

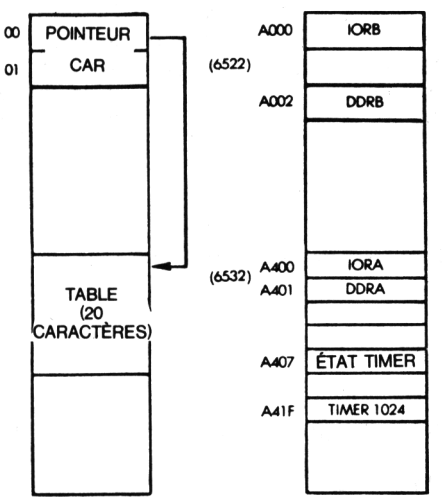


Fig. 6-21 : Carte d'implantation mémoire pour l'imprimante

```
0200  A9  3F        LINE  LDA  #$3F     Configure Port A
   2  8D  01  A4          STA  DDRA
   5  A0  01              LDY  #1       Envoie signal de départ
   7  8C  00  A0          STY  IORB
   A  88                  DEY
   B  8C  00  A0          STY  IORB     Sort « 0 »
   E  2C  00  A4  TST1    BIT  IORA     Lit état
0211  70  FB              BVS  TST1     Demande caractère ?
   3  B1  00              LDA  ($00),Y  Charge caractère
   5  8D  00  A4          STA  IORA     L'imprime
   8  2C  00  A4  TST2    BIT  IORA     Teste l'état
   B  50  FB              BVC  TST2
   D  C8                  INY           Caractère suivant
   E  C0  14              CPY  #$14     20e ?
0220  D0  EC              BNE  TST1
   2  A9  20              LDA  #$20     Caractère espace
   4  8D  00  A4          STA  IORA
   7  A9  30              LDA  #$30     Constante de délai
   9  8D  1F  A4          STA  T1024    Tempo × 1024
   C  2C  07  A4  TTIM    BIT  TIMFLG   Etat du temporisateur
   F  10  FB              BPL  TTIM
0231  60                  RTS
0000  50  00              WORD BUFFER
0050  30  31 32  33 34  BUFFER BYTE '0, '1, '2, '3, '4, '5, '6, '7,
      35 36  37 38  39 40                '8, '9, 'W, 'A, 'B, 'C, 'D,
      41 42  43 44  45 46                'E, 'F, 'G, 'H, 'I
      47 48  49
```

Fig. 6-22 : Programme pour l'imprimante (Programme 6-3)

On commencera par initialiser le registre direction A :

```
LINE    LDA   #$3F
        STA   DDRA
```

Avant de générer l'impulsion de départ, en déposant la valeur "00000001" dans le registre IORB :

```
        LDY   #1          "00000001"
        STY   IORB
```

On remettra, ensuite, IORB à 0 :

```
        DEY                 Y = "00000000"
        STY   IORB
```

Il s'agira, maintenant, de tester la ligne "demande de caractère". On bouclera si elle est à un. Lorsqu'elle deviendra "0", nous irons chercher le caractère suivant :

```
TST 1   BIT   IORA          LECTURE DE L'ÉTAT
        BVS   TST1
```

On se souvient que l'instruction BIT teste un emplacement mémoire, sans perturber son contenu. Elle copie les bits 6 et 7, respectivement dans les indicateurs V et N. Nous nous intéresserons, ici, au bit 6 (se reporter au branchement de l'imprimante, fig. 6-19). L'instruction BVS va tester la valeur de l'indicateur de débordement V, qui a pris la valeur du bit 6 de IORA. On trouvera donc cette valeur sur la ligne "demande caractère", et on obtiendra le caractère suivant dans la table de 20 caractères, rangée à l'adresse contenue dans les cases "00" et "01". Une instruction d'accès indirect aboutirait au premier élément de la table. Or, nous voulons que ce segment de programme soit capable de retrouver n'importe quel élément. Comme dans toute table, on utilisera pour cela une *indexation,* en ayant, en outre, recours au registre index Y, qui contient, au départ, la valeur "00", et sera incrémenté jusqu'à "19", avant la sortie de la boucle. Il s'agit, en l'occurrence, d'une technique d'adressage indirect indexé :

```
        LDA   ($00), Y
```

L'accès indirect indexé est illustré sur la figure 6-23. Nous accéderons d'abord au contenu des adresses 00 et 01, qui sera utilisé comme adresse de base de la table à parcourir. Nous y ajouterons le contenu du registre Y, pour obtenir l'adresse définitive de la donnée recherchée (cf. fig. 6-21). Cette donnée est le code ASCII du caractère à imprimer. Il est envoyé sur IORA.

```
        STA   IORA
```

Lorsque le caractère a été envoyé, il faut attendre que la ligne

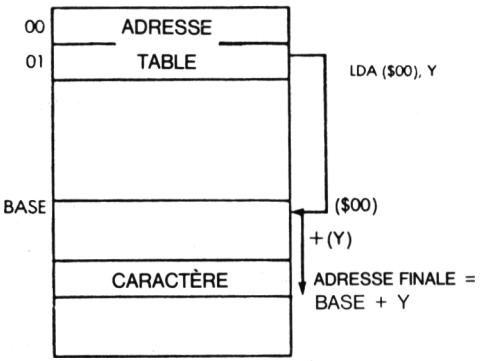


Fig. 6-23 : Adressage indirect indexé

"demande caractère" revienne à 1, au moyen d'une boucle de deux instructions, exactement comme ci-dessous :

```
TST2    BIT   IORA
        BVC   TST2
```

On incrémente, ensuite, le compteur de caractères (registre Y) :

```
        INY
```

et on le teste par rapport à la valeur limite 20 (décimal) = 14 (hexadécimal). Tant que la valeur limite n'est pas atteinte, il faut recommencer la boucle :

```
        CPY   #$14
        BNE   TST1
```

Le code correspondant au caractère « espace » est envoyé sur IORA :

```
        LDA   #$20
        STA   IORA
```

Un délai minimum entre deux lignes successives doit être garanti, par l'intermédiaire du temporisateur, pourvu du facteur

1 024. Le délai de 48 ms est obtenu en chargeant simplement l'adresse-mémoire convenable (cf. fig. 6-19 pour la carte d'implantation-mémoire) :

```
        LDA   #$30        48 EN DECIMAL
        STA   T1024
```

On testera alors, continuellement, l'indicateur d'état du temporisateur jusqu'à ce qu'il devienne "1", et dénotera ainsi le time-out :

```
TTIM    BIT   TIMFLG
        BPL   TTIM
```

L'impression exacte, obtenue pour la ligne-exemple de 20 caractères marquée dans le programme, apparaît figure 6-24 :

```
0123456789@ABCDEFGHI
```

Fig. 6-24 : Impression obtenue

Exercice 6-10 : *Connectez, en même temps, l'imprimante et le lecteur de ruban perforé. Vous devrez imprimer le contenu du ruban perforé à la fin de chaque ligne de 20 caractères.*

RÉCAPITULATION

Nous venons donc d'interfacer, à la carte micro-ordinateur, des périphériques réels, tant du point de vue hardware que software. Nous avons utilisé à fond les possibilités spécifiques des PIO et des modes d'adressage fournis par le 6502, afin d'optimiser les programmes. Le lecteur devrait maintenant avoir acquis toutes les connaissances requises pour réaliser ses propres programmes d'applications, dans la plupart des cas habituels.

7

CONCLUSIONS

Ce livre a présenté, systématiquement, les techniques hardware et software nécessaires au branchement d'une véritable carte 6502 sur le monde réel. Nous avons d'abord décrit les boîtiers d'entrées-sorties, ainsi que les cartes 6502, les plus répandues, puis, dans les chapitres IV, V et VI, présenté des programmes de complexité croissante. Le lecteur ne devrait, maintenant, plus avoir de doutes sur sa capacité à connecter sa propre carte 6502 à des organes d'entrée-sortie usuels, et à résoudre les problèmes d'interface hardware et software liés à cette connexion. En fait, l'auteur estime que les connaissances acquises devraient mettre le lecteur en position de résoudre n'importe quels problèmes d'application de complexité moyenne. Des problèmes d'interface spécifiques peuvent, naturellement, se présenter. Le lecteur est engagé à consulter, à cet effet, la référence *C5 : Techniques d'interface aux microprocesseurs*. Le lecteur qui aurait, jusqu'à présent, négligé de résoudre les exercices proposés aux chapitres IV, V et VI, est vivement encouragé à y revenir. D'abord sur le papier, puis sur une carte micro-ordinateur réelle.

La prochaine étape

Si vous n'avez pas encore construit de carte d'application, la prochaine étape sera, logiquement, de vous rendre dans le magasin d'électronique le plus proche, et d'acheter les quelques composants bon marché nécessaires à la mise en œuvre des applications proposées. Vous devrez, ensuite, vous efforcer d'écrire, par vous-

même, quelques programmes, sans consulter ce livre, afin de vous assurer que vous avez acquis les capacités requises. Faites appel à votre imagination : vous pouvez inventer beaucoup d'autres applications utilisant le même hardware réduit, ou d'autres dispositifs.

Le lecteur intéressé par les techniques de programmation plus complexes pourra se reporter à un troisième volume, à paraître dans cette série, intitulé « Jeux avec le 6502 ». Nous y introduirons et décrirons des algorithmes plus élaborés, grâce auxquels le lecteur pourra pratiquer des jeux variés, tels que le « casse-tête », ou les carrés magiques. Le hardware requis pour ces jeux est minimum : un clavier 16 touches, 15 LED et un haut-parleur.

Le temps d'apprentissage de la programmation varie considérablement d'une personne à l'autre. Il reste que l'étape qui suit logiquement la lecture de n'importe quel livre de programmation est toujours la même : la pratique. Nous espérons que le contenu de ce livre vous aura apporté les connaissances susceptibles de vous aider dans cette voie.

APPENDICE A

UN ASSEMBLEUR 6502 EN BASIC

INTRODUCTION

Il est possible de développer sur le papier de courts programmes pour 6502, puis de les faire entrer dans le micro-ordinateur. Néanmoins, dans le cas de programmes plus longs (disons, plus de quelques dizaines d'instructions), ou s'il s'agit de développer un grand nombre de petits programmes, il devient commode d'avoir recours à un assembleur. Il est probable que la plupart des lecteurs intéressés sérieusement à traiter des applications réelles à l'aide du 6502, seront amenés à développer de tels programmes. Ce livre comprend donc le listing complet d'un assembleur pour le 6502, écrit en BASIC à l'intention de ceux qui n'y auraient pas encore eu accès.

L'avantage d'un assembleur pour 6502, écrit en BASIC, est qu'il peut fonctionner sur n'importe quel ordinateur possédant BASIC et accessible au lecteur. La version de BASIC, utilisée ici, est celle des ordinateurs Hewlett-Packard. Elle peut être considérée comme un sous-ensemble des BASIC de la plupart des micro-ordinateurs, en ce sens qu'elle est dépourvue d'un certain nombre des possibilités qu'on trouve dans ces derniers. L'utilisation de cet assembleur sur un ordinateur muni d'un autre BASIC nécessite une traduction. Mais les difficultés ne devraient être nullement insurmontables, puisque les BASIC les plus fréquemment disponibles sur des micro-ordinateurs disposent de beaucoup plus de possibilités que celui utilisé ici. Notre assembleur est essentiellement compatible vers le

haut. En fait, un bon programmeur, connaissant bien son BASIC, sera probablement capable de réduire considérablement le nombre d'instructions.

Cet assembleur a été utilisé, de façon satisfaisante, dans un grand nombre de programmes à l'usage du 6502. Il est donc à notre connaissance fiable. Présenté ici à des fins pédagogiques, il ne saurait toutefois, être garanti, en aucune façon. Une version en BASIC Microsoft sera publiée prochainement.

On trouvera, ci-dessous, un exemple de sortie démontrant le fonctionnement de l'assembleur.

Tous les programmes de la fin du chapitre IV ont été assemblés de cette façon.

DESCRIPTION GÉNÉRALE

ASM65 est un assembleur complet des mnémoniques du 6502. Il reconnaît tous les mnémoniques standards de l'industrie, et produit le listing hexadécimal standard, tel que le montre la figure A-1.

Il admet, en outre, les directives standards de l'industrie, à cette seule exception qu'il utilise un "." au lieu d' "*", pour désigner l'adresse courante de PC. Les directives disponibles sont .BYT, .WORD, .DBYT, .TEXT. L'utilisateur se reportera pour le détail de ces directives à la description de l'assembleur de n'importe quel constructeur.

UTILISATION DE L'ASSEMBLEUR

ASM65 est écrit en BASIC F des Hewlett-Packard, série 2000. On trouvera, plus loin, une description des caractéristiques de cette version de BASIC. Adapter cet interpréteur à d'autres versions éventuelles de BASIC, ne devrait entraîner que peu de modifications.

ASM 65 opère sur des fichiers séquentiels : trois au minimum, et le plus souvent quatre. Ce sont : le fichier source, le fichier de la table des symboles, un fichier temporaire et, sur option, un fichier destination, différent du fichier source.

Le fichier d'entrée contient les instructions en langage assembleur.

```
% CAT SRC
;MEMORY BLOCK MOVE PROGRAM
;MOVES UP TO 255 BYTES FROM A TABLE STARTING AT
;LOC1 TO A TABLE STARTING AT LOC2. LENGTH OF THE
;SECTION TO BE MOVED IS IN MOVLEN.
MOVLEN  =$00
LOC1    =$200
LOC2    =$300
;
        LDX MOVLEN ;LOAD LENGTH OF MOVE TO INDEX
LOOP    LDA LOC1,X ;LOAD BYTE TO BE MOVED
        STA LOC2,X ;STORE BYTE TO BE MOVED
        DEX ;COUNT DOWN
        BPL LOOP ;IF NOT DONE, MOVE NEXT BYTE
        RTS ;DONE
% RUN ASM65
SOURCE FILE ?SRC
OBJECT FILE ?DEST
PRINTOUT ?YES
ASSEMBLY BEGINS...
                ;MEMORY BLOCK MOVE PROGRAM
                ;MOVES UP TO 255 BYTES FROM A TABLE STARTING AT
                ;LOC1 TO A TABLE STARTING AT LOC2. LENGTH OF THE
                ;SECTION TO BE MOVED IS IN MOVLEN.
                MOVLEN  =$00
                LOC1    =$200
                LOC2    =$300
                ;
0000: A6 00             LDX MOVLEN    ;LOAD LENGTH OF MOVE TO INDEX
0002: BD 00 02  LOOP    LDA LOC1,X    ;LOAD BYTE TO BE MOVED
0005: 9D 00 03          STA LOC2,X    ;STORE BYTE TO BE MOVED
0008: CA                DEX           ;COUNT DOWN
0009: 10 F7             BPL LOOP      ;IF NOT DONE, MOVE NEXT BYTE
000B: 60                RTS           ;DONE


SYMBOL TABLE:
 MOVLEN      0000        LOC1        0200        LOC2        0300
 LOOP        0002
DONE
```

Fig. A1 : Utilisation de l'assembleur ASM 65

Il doit donc contenir du texte ASCII, et être structuré selon les règles de la syntaxe de l'assembleur (décrites section suivante). En général, les lignes d'entrée peuvent être écrites en format libre, avec des champs séparés par un ou plusieurs espaces. Néanmoins, toute étiquette doit commencer en colonne 1. Toute ligne sans étiquette doit commencer autre part qu'en colonne 1.

L'assembleur met automatiquement au format le champ commentaire sur le fichier de sortie, mais pas les autres champs de l'instruction. De sorte que l'utilisateur peut tabuler ses instructions en entrée, de façon raisonnable, pour obtenir un listing clair. L'opération a pour but d'améliorer la lisibilité.

Le fichier de sortie est, lui aussi, constitué de texte ASCII, y compris la représentation de tous les nombres. Le fichier de sortie peut, sur option, être imprimé, après exécution de la seconde passe de l'assembleur. Un message est imprimé sur le listing, ou affiché sur l'écran : "PRINTOUT ?". L'utilisateur répond "YES", s'il veut la sortie. "NO", dans le cas contraire.

L'assembleur fournit des diagnostics détaillés, décrit toutes les erreurs décelées, puis les liste sur la sortie.

Le message d'erreur peut contenir différents marqueurs de zones. Par exemple les délimiteurs d'opérateurs ("!") et le délimiteur de référence interne non résolue ("***").

La table des symboles donne, comme d'habitude, l'adresse hexadécimale de toutes les étiquettes symboliques utilisées par le programme. La figure A-2 en montre un exemple.

```
SYMBOL TABLE:
 MOVLEN        0000          LOC1          0200          LOC2          0300
 LOOP          0002
DONE
```

Fig. A-2 : La table des symboles

SYNTAXE

Constantes

Les constantes peuvent être exprimées dans l'une des quatre représentations habituelles :

- Hexadécimal : la constante doit être précédée par un "$". Exemple : "LDA $20" charge l'accumulateur, à partir de l'adresse mémoire "20", en hexadécimal.
- Binaire : elle doit être précédée par un "%". Exemple : "LDA #%11111111" charge l'accumulateur avec uniquement des uns.
- Décimal : représentation habituelle. Exemple : "LDA #0", charge l'accumulateur avec la valeur décimale 0.
- ASCII : doit être précédée par une apostrophe (« ' »). Exemple : "LDA #'A" charge l'accumulateur avec le code ASCII de la lettre A.

Expressions arithmétiques

Il est possible d'utiliser des expressions arithmétiques dans un champ opérande, dans une assignation d'étiquette ou dans une instruction d'allocation-mémoire.

L'opérande, dans une expression arithmétique, peut être soit un nombre exprimé dans n'importe quelle représentation, soit une étiquette, soit un "." (symbole de l'adresse courante) soit encore toute combinaison des précédents. Les opérateurs autorisés sont "+" et "−". Dans le cas où on utilise plus d'un opérateur, l'expression arithmétique est évaluée de gauche à droite.

Commentaires

Les commentaires doivent être précédés de ";". Ils peuvent commencer en n'importe quelle colonne, même en colonne 1. Tous les commentaires seront justifiés au milieu de la feuille d'impression, sauf s'ils commencent en colonne 1.

Assignation de mémoire

Les allocations de mémoire sont effectuées par les quatre directives :

.BYT — Assigne un octet de donnée à un emplacement-mémoire.

.WORD — Assigne deux octets de donnée à deux emplacements-mémoire consécutifs, en commençant par l'octet bas.

.DBYT — Assigne deux octets de donnée à deux emplacements-mémoire consécutifs, en commençant par l'octet haut.

.TEXT — Convertit une chaîne de caractères ASCII en données hexa, et la range à des adresses consécutives. La chaîne doit être délimitée par deux caractères identiques, non blancs.

Il n'y a pas de directive de fin : une fin de fichier en tient lieu.

Exemple d'une allocation-mémoire :

```
.BYT    $2A, CONSTANTE
.WORD   2,%10
```

LE BASIC FHP 2000

,Le BASIC de Hewlett-Packard est différent de beaucoup d'autres BASIC utilisés sur des mini-ordinateurs ou micro-ordinateurs courants. Il s'adapte toutefois facilement. La suite est la liste d'un certain nombre de caractéristiques différentes de la plupart des BASIC, ou du standard de Dartmouth.

Fichiers

Les fichiers sont déclarés au début du programme dans une instruction FILES. Ils sont numérotés par ordre d'apparition. L'instruction ASSIGN assigne un fichier, spécifié par son premier argument, à un numéro de fichier, spécifié par le second argument. Le troisième argument est une variable muette. Une étoile dans l'instruction FILES désigne un fichier, qui sera plus tard assigné à cette position par une instruction ASSIGN. L'instruction READ lit le fichier. Son premier argument, précédé de " # " est le numéro du fichier. Si le numéro d'enregistrement est 1, et s'il n'y a pas de ";", l'instruction est utilisée pour remettre à 0 le pointeur du fichier, comme dans : "READ # 2, 1". Tous les arguments, après le ";", sont les variables à lire.

L'instruction PRINT est semblable au READ. Elle comporte, elle aussi, une forme spéciale "PRINT # 2, END", qui écrit une marque de fin de fichier.

L'instruction IF END # THEN agit de façon analogue à une interruption vectorisée. Lorsqu'on trouve une fin de fichier en lecture, le programme saute au numéro de ligne cité après THEN, évitant ainsi un arrêt irrémédiable. Cette opération a lieu, même si le IF END n'est pas en cours d'exécution. Autrement dit, le vecteur n'a besoin d'être spécifié qu'une fois, sauf s'il faut le changer.

Chaînes de caractères

Les chaînes sont uni-dimensionnelles, et ne peuvent être dimensionnées qu'ainsi. Pour assigner à une chaîne la longueur zéro (= la vider), on emploie une instruction du type : L$ = " ". Les caractères d'une chaîne sont référencés comme suit : pour référencer la sous-chaîne d'une chaîne plus grande, on utilise la forme T$(*a*,*b*), où *a* et *b* sont des expressions spécifiant, respectivement le numéro du premier et du dernier caractère de la sous-chaîne désirée. Les caractères d'une chaîne sont numérotés de gauche à droite, en commençant à 1. L'exemple : si A$ = "ABCDE" et si on fait B$ = A$(2,3), B$ sera la chaîne "BC".

La forme T$(*a*) référence tous les caractères de T$, depuis le a^e jusqu'à la fin de T$. Exemple : si A$ = "12345", A$(3) est la sous-chaîne "345".

Les fonctions chaînes CHR$ et ASC$, qui convertissent, respectivement, un nombre décimal en la chaîne de 1 caractère dont il est le code ASCII, et une chaîne de 1 caractère en son code ASCII, ne sont pas disponibles. C'est pourquoi ASM65 lit une chaîne de caractères ordonnés, suivant leur code ASCII, sur un fichier système appelé $ASCIIF, pour effectuer de telles conversions.

MAX renvoie le maximum de deux valeurs. Par exemple : B = 11 MAX 9 donne à B la valeur 11. MIN renvoie le minimum de deux valeurs. Dans une instruction PRINT, LIN *x* fait passer *x* lignes.

Les définitions ci-dessus n'ont d'autre but que de servir de guide à la traduction de ASM65 en d'autres versions de BASIC.

Fig. A-3 : Listing de l'Assembleur 6502
copyright © 1979, Sybex Inc.

```
ASM65

10  REM : ********** 6502 MNEMONIC ASSEMBLER, VERSION 2.0 **********
20  REM
30  REM : WRITTEN IN HP2000F TSS BASIC.
40  REM : CAN BE USED WITH ALL 65XX PROCESSORS AS MADE BY COMMODORE,
50  REM : SYNERTEK, AND ROCKWELL.
60  REM : ALL MNEMONICS AND DIRECTIVES ARE INDUSTRY STANDARD, WITH
70  REM : THE EXEPTION OF THE USE OF '.' FOR CURRENT ADDRESS.
80  R=10
90  T9=0
100  A=0
110  DIM L$[72],M$[72],O$[72],C$[72],Z$[72],P$[72],T$[72]
120  DIM A$[72],N$[72]
130  DIM I$[72]
140  L=0
150  FILES *,SYMTAB,TEMP,*,$ASCIIF
160  PRINT "SOURCE FILE ";
170  INPUT T$
180  PRINT "OBJECT FILE ";
190  INPUT O$
200  ASSIGN T$,1,Q8
210  ASSIGN O$,4,Q8
220  READ #1,1
230  PRINT #2,1
240  PRINT #3,1
250  R8=0
260  PRINT "PRINTOUT ";
270  INPUT I$
280  IF I$ <> "NO" THEN 300
290  R8=1
300  PRINT "ASSEMBLY BEGINS..."
310  C=0
```

```
320  IF  END #1 THEN 2440
330  L$=""
340  I$=""
350  M$=""
360  O$=""
370  C$=""
380  Z$="
390  L=L+1
400  REM********** SEPARATE TOKENS, STORE LABEL ASSIGNMENTS **********
410  READ #1;I$
420  T5=C
430  IF I$="" THEN 830
440  P=1
450  P$=";"
460  GOSUB 3970
470  IF P1=0 THEN 510
480  IF P1=1 THEN 800
490  C$=I$[P1]
500  I$=I$[1,P1-1]
510  IF I$[1,1]=" " THEN 590
520  GOSUB 3790
530  L$=P$
540  IF L$ <> "." THEN 590
550  M$="."
560  GOSUB 4940
570  L$=""
580  GOTO 860
590  GOSUB 3790
600  M$=P$
610  IF M$[1,3]=".WO" THEN 3110
620  IF M$[1,3]=".TE" THEN 3110
630  IF M$[1,3]=".BY" THEN 3110
640  IF M$[1,3]=".DB" THEN 3110
650  IF M$ <> "" THEN 850
660  C$=C$[1,34]
670  IF LEN(L$) <> 0 THEN 700
680  I$=I$[1,19]
690  GOTO 820
700  GOSUB 3790
710  N$=P$
720  IF LEN(N$) <> 0 THEN 750
730  T1=C
740  GOTO 780
750  GOSUB 4070
760  IF T4=2 THEN 830
770  T1=F1
780  PRINT #2;L$,T1
790  PRINT #2; END
800  I$=I$[1,LEN(I$) MIN 55]
810  Z$[17,17+LEN(I$)]=I$
820  Z$[(LEN(I$)+19 MAX 38) MIN 72]=C$
830  PRINT #3;Z$,T5
840  GOTO 320
850  IF M$[1,1] <> "." THEN 1050
860  P$="="
870  GOSUB 3970
880  IF P1>0 THEN 910
890  PRINT "MISSING '=' IN LINE ";L
900  GOTO 3090
910  P=P1+1
920  GOSUB 3790
930  IF P$[1,1] <> "" THEN 960
940  PRINT "MISSING ARGUMENT IN LINE ";L
950  GOTO 3090
960  N$=P$
```

```
970  GOSUB 4070
980  IF T4 <> 2 THEN 1010
990  PRINT "ILLEGAL FORWARD REFERENCE IN LINE ";L
1000  GOTO 3090
1010  T1=C
1020  C=F1
1030  IF L$ <> "" THEN 780
1040  GOTO 800
1050  RESTORE 5710
1060  IF M$="" THEN 1140
1070  FOR I=1 TO 56
1080  READ T$
1090  IF T$=M$ THEN 1130
1100  NEXT I
1110  PRINT "UNKNOWN OPCODE IN LINE ";L
1120  GOTO 3090
1130  O=I
1140  IF L$="" THEN 1170
1150  PRINT #2;L$,C
1160  PRINT #2; END
1170  GOSUB 3750
1180  O$=P$
1190  I$[P-LEN(O$)-1,P-LEN(O$)-1]="!"
1200  REM********** FIND ADDRESSING MODES, LOAD EFFECTIVE ADDRESS **********
1210  IF O$ <> "" THEN 1240
1220  M=1
1230  GOTO 2200
1240  IF O$ <> "A" THEN 1270
1250  M=2
1260  GOTO 2200
1270  IF O$[1,1] <> "#" THEN 1320
1280  M=3
1290  P=P+1
1300  N$=O$[2]
1310  GOTO 1870
1320  IF M$[1,1] <> "B" THEN 1460
1330  IF M$="BIT" THEN 1460
1340  M=12
1350  N$=O$
1360  GOSUB 4070
1370  IF T4 <> 2 THEN 1400
1380  A=-200
1390  GOTO 1970
1400  A=F1-C-2
1410  IF A >= 0 THEN 1430
1420  A=256+A
1430  IF ABS(F1-C) <= 127 THEN 1970
1440  PRINT "BRANCH OUT OF RANGE IN LINE ";L
1450  GOTO 3090
1460  P$="("
1470  P=P-LEN(O$)
1480  GOSUB 3970
1490  P5=P1
1500  P$=","
1510  GOSUB 3970
1520  P6=P1
1530  P7=0
1540  IF  NOT P6 THEN 1610
1550  IF I$[P6+1,P6+1] <> "X" THEN 1580
1560  P7=1
1570  GOTO 1610
1580  IF I$[P6+1,P6+1]="Y" THEN 1610
1590  PRINT "BAD ADDRESSING MODE IN LINE ";L
1600  GOTO 3090
1610  IF P5 <> 0 THEN 1780
```

```
1620  GOSUB 3790
1630  N$=P$
1640  IF  NOT P6 OR  NOT P7 THEN 1670
1650  M=5
1660  GOTO 1710
1670  IF  NOT P6 THEN 1700
1680  M=6
1690  GOTO 1710
1700  M=4
1710  GOSUB 4070
1720  A=F1
1730  IF T4 <> 2 THEN 1750
1740  A=-1000
1750  IF ABS(A) <= 255 THEN 1970
1760  M=M+3
1770  GOTO 1970
1780  GOSUB 3790
1790  N$=P$[2]
1800  IF  NOT P6 OR  NOT P7 THEN 1830
1810  M=10
1820  GOTO 1870
1830  IF  NOT P6 THEN 1860
1840  M=11
1850  GOTO 1870
1860  M=13
1870  GOSUB 4070
1880  A=F1
1890  IF (M <> 10 AND M <> 11) OR A <= 255 THEN 1920
1900  PRINT "VALUE TOO LARGE FOR ZERO PAGE IN LINE ";L
1910  GOTO 3090
1920  IF T4 <> 2 THEN 1970
1930  A=-1000
1940  IF M=13 THEN 1970
1950  A=-200
1960  REM********** PRINT OPCODES & EA ON FILE **********
1970  IF A >= 0 THEN 2070
1980  Z$[10,11]="**"
1990  C=C+1
2000  IF M <> 12 THEN 2020
2010  Z$[11,11]="R"
2020  W9=A+256
2030  IF W9 >= 0 THEN 2200
2040  Z$[13,14]="**"
2050  C=C+1
2060  GOTO 2200
2070  R=16
2080  I=A
2090  GOSUB 4940
2100  T$=A$
2110  A$="000"
2120  A$[4]=T$
2130  IF (M >= 3 AND M <= 6) OR (M >= 10 AND M <= 12) THEN 2180
2140  Z$[13,14]=A$[LEN(A$)-3,LEN(A$)-2]
2150  Z$[10,11]=A$[LEN(A$)-1]
2160  C=C+2
2170  GOTO 2200
2180  Z$[10,11]=A$[LEN(A$)-1]
2190  C=C+1
2200  R=16
2210  I=T5
2220  GOSUB 4940
2230  T$="000"
2240  T$[4]=A$
2250  Z$[1,4]=T$[LEN(T$)-3]
2260  RESTORE 5140
```

```
2270  FOR I=1 TO (O-1)*13+M
2280  READ T$
2290  NEXT I
2300  IF T$ <> "   " THEN 2370
2310  IF M>6 OR M<4 THEN 2350
2320  M=M+3
2330  C=T5
2340  GOTO 1970
2350  PRINT "ILLEGAL ADDRESSING MODE IN LINE ";L
2360  GOTO 3090
2370  Z$[7,8]=T$
2380  Z$[5,5]=":"
2390  C=C+1
2400  Z$[17,17+LEN(I$)]=I$
2410  Z$[(19+LEN(I$)) MAX 38]=C$[1,72-(19+LEN(I$) MAX 38)]
2420  PRINT #3;Z$,T5
2430  GOTO 320
2440  REM********** SECOND PASS: RESOLVE FWD REFERENCES **********
2450  PRINT #2; END
2460  PRINT #3; END
2470  READ #2,1
2480  L=0
2490  READ #3,1
2500  PRINT #4,1
2510  IF  END #3 THEN 2870
2520  P=1
2530  READ #3;I$,T5
2540  L=L+1
2550  IF I$="" THEN 2850
2560  P$="!"
2570  GOSUB 3970
2580  IF P1=0 OR P1=17 THEN 2610
2590  P=P1
2600  I$[P,P]=" "
2610  IF I$[10,10] <> "*" THEN 2850
2620  GOSUB 3790
2630  N$=P$
2640  IF N$[1,1] <> "(" THEN 2660
2650  N$=N$[2]
2660  GOSUB 4070
2670  IF T4 <> 2 THEN 2700
2680  PRINT "IRRESOLVABLE FWD REF / BAD LABEL IN LINE ";L
2690  GOTO 3090
2700  I=F1
2710  IF I$[11,11] <> "R" THEN 2750
2720  I=F1-T5-2
2730  IF I >= 0 THEN 2750
2740  I=I+256
2750  R=16
2760  GOSUB 4940
2770  T$=A$
2780  A$="000"
2790  A$[4]=T$
2800  IF I$[13,14] <> "**" THEN 2840
2810  I$[13,14]=A$[LEN(A$)-3,LEN(A$)-1]
2820  I$[10,11]=A$[LEN(A$)-1]
2830  GOTO 2850
2840  I$[10,11]=A$[LEN(A$)-1]
2850  PRINT #4;I$
2860  GOTO 2510
2870  PRINT #4; END
2880  IF R8=1 THEN 3080
2890  IF  END #4 THEN 2940
2900  READ #4,1
2910  READ #4;I$
```

```
2920  PRINT I$
2930  GOTO 2910
2940  READ #2,1
2950  PRINT LIN(2);"SYMBOL TABLE:"
2960  IF  END #2 THEN 3080
2970  FOR I6=1 TO 3
2980  READ #2;O$,T5
2990  R=16
3000  I=T5
3010  GOSUB 4940
3020  T$="0000"
3030  T$[LEN(T$)+1]=A$
3040  PRINT TAB((I6-1)*25+1);O$;TAB((I6-1)*25+13);T$[LEN(T$)-3];
3050  NEXT I6
3060  PRINT
3070  GOTO 2970
3080  END
3090  PRINT "<"I$">"
3100  END
3110  REM********** PROCESS MEMORY LOADS **********
3120  Q7=1
3130  IF M$[2,3] <> "TE" THEN 3260
3140  IF Q7 <> 1 THEN 3190
3150  GOSUB 3750
3160  P=P-LEN(P$)
3170  O$=I$[P,P]
3180  P=P+1
3190  IF P <= 72 THEN 3220
3200  PRINT "BAD DELIMITER IN LINE ";L
3210  GOTO 3090
3220  P$[1]=""
3230  P$[2,2]=I$[P,P]
3240  IF P$[2,2]=O$ THEN 320
3250  GOTO 3280
3260  GOSUB 3790
3270  Z$=""
3280  P=P+1
3290  IF LEN(P$)=0 THEN 320
3300  N$=P$
3310  GOSUB 4070
3320  IF T4 <> 2 THEN 3350
3330  PRINT "BAD LABEL IN MEMORY ASSIGNMENT OF LINE ";L
3340  GOTO 3090
3350  R=16
3360  I=F1
3370  GOSUB 4940
3380  T$=A$
3390  A$="000"
3400  A$[4]=T$
3410  IF M$[2,2] <> "W" THEN 3460
3420  Z$[10,11]=A$[LEN(A$)-3,LEN(A$)-2]
3430  Z$[7,8]=A$[LEN(A$)-1]
3440  C=C+2
3450  GOTO 3560
3460  IF M$[2,2]="D" THEN 3530
3470  IF F1<256 THEN 3500
3480  PRINT "NUMBER TOO LARGE IN MEMORY ASSIGNMENT OF LINE ";L
3490  GOTO 3090
3500  Z$[7,8]=A$[LEN(A$)-1]
3510  C=C+1
3520  GOTO 3560
3530  Z$[7,8]=A$[LEN(A$)-3,LEN(A$)-2]
3540  Z$[10,11]=A$[LEN(A$)-1]
3550  C=C+1
3560  I=T5
```

```
3570  R=16
3580  GOSUB 4940
3590  T$="000"
3600  T$[4]=A$
3610  Z$[1,4]=T$[LEN(T$)-3]
3620  Z$[5,5]=":"
3630  IF Q7 <> 1 THEN 3700
3640  IF LEN(L$)=0 THEN 3670
3650  PRINT #2;L$,T5
3660  PRINT #2; END
3670  Z$[17,17+LEN(I$)]=I$
3680  Z$[(19+LEN(I$)) MAX 38]=C$[1,72-(19+LEN(I$)) MAX 38]
3690  GOTO 3710
3700  Z$=Z$[1,15]
3710  Q7=0
3720  PRINT #3;Z$,T5
3730  T5=C
3740  GOTO 3130
3750  REM ***** ROUTINE TO ISOLATE TOKEN *****
3760  REM : STARTS LOOKING FOR TOKEN AT P, PUTS IT IN P$, AND
3770  REM : UPDATES P. IF ENTERED HERE, STOPS SCAN AT ' '.
3780  T9=1
3790  REM : IF ENTERED HERE, STOPS SCAN AT ' ', ',', ')', '='.
3800  FOR I1=P TO LEN(I$)
3810  IF I$[I1,I1] <> " " THEN 3830
3820  NEXT I1
3830  P$=""
3840  FOR I2=I1 TO LEN(I$)
3850  IF I$[I2,I2]=" " THEN 3920
3860  IF T9=1 THEN 3900
3870  IF I$[I2,I2]="," THEN 3920
3880  IF I$[I2,I2]=")" THEN 3920
3890  IF I$[I2,I2]="=" THEN 3920
3900  P$[LEN(P$)+1]=I$[I2,I2]
3910  NEXT I2
3920  P=I2
3930  IF LEN(P$) <> 0 THEN 3950
3940  P=P+1
3950  T9=0
3960  RETURN
3970  REM ***** FIND SYMBOL ROUTINE *****
3980  REM : RETURNS P1=SYMLOC IF IT IS FOUND, P1=0
3990  REM : IF SYMBOL NOT FOUND
4000  FOR I=P TO LEN(I$)
4010  IF I$[I,I]=P$[1,1] THEN 4050
4020  NEXT I
4030  P1=0
4040  RETURN
4050  P1=I
4060  RETURN
4070  REM ***** NUMERIC STRING INTERPRETER *****
4080  REM : SIMPLIFIES STRINGS OF LABELS AND NUMERIC EXPRESSIONS
4090  REM : OF NUMBERS IN ANY BASE, PLUS ASCII CONSTANTS.
4100  F1=W=0
4110  A$=""
4120  FOR I=1 TO LEN(N$)
4130  IF N$[I,I]="+" THEN 4180
4140  IF N$[I,I]="-" THEN 4180
4150  IF N$[I,I]=")" THEN 4610
4160  A$[LEN(A$)+1]=N$[I,I]
4170  NEXT I
4180  IF A$ <> "." THEN 4210
4190  F2=C
4200  GOTO 4480
4210  IF A$[1,1]>"Z" THEN 4350
4220  IF A$[1,1]<"A" THEN 4350
```

```
4230  READ #2,1
4240  IF  END #2 THEN 4330
4250  READ #2;T$,T1
4260  IF T$ <> A$ THEN 4240
4270  F2=T1
4280  T4=3
4290  IF  END #2 THEN 4320
4300  READ #2;T$,T1
4310  GOTO 4300
4320  GOTO 4480
4330  T4=2
4340  RETURN
4350  IF A$[1,1] <> "/" THEN 4390
4360  A$=A$[2]
4370  GOSUB 4640
4380  GOTO 4480
4390  B=10
4400  IF A$[1,1] <> "%" THEN 4430
4410  B=2
4420  GOTO 4450
4430  IF A$[1,1] <> "$" THEN 4460
4440  B=16
4450  A$=A$[2]
4460  GOSUB 4750
4470  F2=F
4480  IF W=2 THEN 4510
4490  F1=F1+F2
4500  GOTO 4520
4510  F1=F1-F2
4520  IF I >= LEN(N$) THEN 4610
4530  T$="+-"
4540  FOR W=1 TO LEN(T$)
4550  IF T$[W,W]=N$[I,I] THEN 4590
4560  NEXT W
4570  PRINT "ILLEGAL OPERATOR IN LINE ";L
4580  GOTO 3090
4590  A$=""
4600  GOTO 4170
4610  T4=0
4620  RETURN
4630  REM ***** ASCII CHARACTER TO NUMBER CONVERTER *****
4640  A$=A$[1,1]
4650  F2=0
4660  READ #5,1
4670  READ #5;T$
4680  FOR I=1 TO 72
4690  IF A$[1,1]=T$[I,I] THEN 4740
4700  F2=F2+1
4710  NEXT I
4720  F2=F2-8
4730  GOTO 4670
4740  RETURN
4750  REM ***** MULTI-RADIX STRING TO NUMBER CONVERTER *****
4760  REM : B IS BASE OF NUMBER IN A$, F IS PRODUCT.
4770  F=0
4780  I1=0
4790  FOR I2=LEN(A$) TO 1 STEP -1
4800  RESTORE 4910
4810  FOR N=0 TO B-1
4820  READ F$
4830  IF F$=A$[I2,I2] THEN 4870
4840  NEXT N
4850  PRINT "BAD NUMBER IN LINE ";L
4860  GOTO 3090
4870  F=F+N*B^I1
```

```
4880  I1=I1+1
4890  NEXT I2
4900  RETURN
4910  DATA "0","1","2","3","4","5","6","7","8","9","A","B","C","D"
4920  DATA "E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S"
4930  DATA "T","U","V","W","X","Y","Z"
4940  REM ***** MULTI-RADIX NUMBER TO STRING CONVERTER
4950  REM : I IS INPUT NUMBER, R IS BASE THAT A$ WILL BE AS PRODUCT.
4960  A$=""
4970  T=I
4980  FOR N=20 TO 0 STEP -1
4990  IF T/R^N >= 1 THEN 5020
5000  NEXT N
5010  N=N-1
5020  Q=INT(T/R^N)
5030  IF Q <= R-1 THEN 5050
5040  Q=0
5050  T=T-Q*R^N
5060  RESTORE 4910
5070  FOR S=0 TO Q
5080  READ T$
5090  NEXT S
5100  A$[LEN(A$)+1]=T$
5110  IF N>0 THEN 5010
5120  RETURN
5130  REM *************** OPCODE TABLE ***************
5140  DATA "  ","  ","69","65","75","  ","6D","7D","79","61","71","  ","  "
5150  DATA "  ","  ","29","25","35","  ","2D","3D","39","21","31","  ","  "
5160  DATA "  ","0A","  ","06","16","  ","0E","1E","  ","  ","  ","  ","  "
5170  DATA "  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","90","  "
5180  DATA "  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","B0","  "
5190  DATA "  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","F0","  "
5200  DATA "  ","  ","  ","24","  ","  ","2C","  ","  ","  ","  ","  ","  "
5210  DATA "  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","30","  "
5220  DATA "  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","D0","  "
5230  DATA "  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","10","  "
5240  DATA "00","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5250  DATA "  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","50","  "
5260  DATA "  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","70","  "
5270  DATA "18","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5280  DATA "D8","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5290  DATA "58","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5300  DATA "B8","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5310  DATA "  ","  ","C9","C5","D5","  ","CD","DD","D9","C1","D1","  ","  "
5320  DATA "  ","  ","E0","E4","  ","  ","EC","  ","  ","  ","  ","  ","  "
5330  DATA "  ","  ","C0","C4","  ","  ","CC","  ","  ","  ","  ","  ","  "
5340  DATA "  ","  ","  ","C6","D6","  ","CE","DE","  ","  ","  ","  ","  "
5350  DATA "CA","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5360  DATA "88","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5370  DATA "  ","  ","49","45","55","  ","4D","5D","59","41","51","  ","  "
5380  DATA "  ","  ","  ","E6","F6","  ","EE","FE","  ","  ","  ","  ","  "
5390  DATA "E8","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5400  DATA "C8","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5410  DATA "  ","  ","  ","  ","  ","  ","4C","  ","  ","  ","  ","  ","6C"
5420  DATA "  ","  ","  ","  ","  ","  ","20","  ","  ","  ","  ","  ","  "
5430  DATA "  ","  ","A9","A5","B5","  ","AD","BD","B9","A1","B1","  ","  "
5440  DATA "  ","  ","A2","A6","  ","B6","AE","  ","BE","  ","  ","  ","  "
5450  DATA "  ","  ","A0","A4","B4","  ","AC","BC","  ","  ","  ","  ","  "
5460  DATA "  ","4A","  ","46","56","  ","4E","5E","  ","  ","  ","  ","  "
5470  DATA "EA","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5480  DATA "  ","  ","09","05","15","  ","0D","1D","19","01","11","  ","  "
5490  DATA "48","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5500  DATA "08","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5510  DATA "68","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5520  DATA "28","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
```

```
5530  DATA "  ","2A","  ","26","36","  ","2E","3E","  ","  ","  ","  ","  "
5540  DATA "  ","6A","  ","66","76","  ","6E","7E","  ","  ","  ","  ","  "
5550  DATA "40","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5560  DATA "60","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5570  DATA "  ","  ","E9","E5","F5","  ","ED","FD","F9","E1","F1","  ","  "
5580  DATA "38","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5590  DATA "F8","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5600  DATA "78","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5610  DATA "  ","  ","  ","85","95","  ","8D","9D","99","81","91","  ","  "
5620  DATA "  ","  ","  ","86","  ","96","8E","  ","  ","  ","  ","  ","  "
5630  DATA "  ","  ","  ","84","94","  ","8C","  ","  ","  ","  ","  ","  "
5640  DATA "AA","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5650  DATA "A8","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5660  DATA "BA","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5670  DATA "8A","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5680  DATA "9A","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5690  DATA "98","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  "
5700  REM *************** MNEMONIC TABLE ***************
5710  DATA "ADC","AND","ASL","BCC","BCS","BEQ","BIT","BMI","BNE"
5720  DATA "BPL","BRK","BVC","BVS","CLC","CLD","CLI","CLV","CMP","CPX","CPY"
5730  DATA "DEC","DEX","DEY","EOR","INC","INX","INY","JMP","JSR","LDA","LDX"
5740  DATA "LDY","LSR","NOP","ORA","PHA","PHP","PLA","PLP","ROL","ROR","RTI"
5750  DATA "RTS","SBC","SEC","SED","SEI","STA","STX","STY","TAX","TAY","TSX"
5760  DATA "TXA","TXS","TYA"
5770  END
```

APPENDICE B

PROGRAMME DU JEU DE MULTIPLICATION

```
                        ;***** MULT *****
                        ;
                        N       =$00
                        P       =$01
                        NSAVE   =$02
                        PSAVE   =$03
                        T       =$04
                        D       =$9D
                        X       =$200
                        Y       =$201
                        RESUL   =$202
                        ASAVE   =$240
                        XSAVE   =$241
                        YSAVE   =$242
                        PA      =$1700
                        PAD     =$1701
                        TIMER   =$1707
                        ;
                                .=$20
0020: A5 00             START   LDA N
0022: 85 02                     STA NSAVE
0024: A5 01                     LDA P
0026: 85 03                     STA PSAVE
0028: A9 01                     LDA #$01
002A: 8D 01 17                  STA PAD
002D: 20 50 02          M1      JSR SOUND
0030: 20 90 00                  JSR DL250
```

```
0033: C6 00             DEC N
0035: D0 F6             BNE M1
0037: A2 14             LDX #$14        ;2 SECOND
0039: 20 9E 00          JSR TIME10      ;.1 SEC SUBROUTINE
003C: 20 50 02  M2      JSR SOUND
003F: 20 90 00          JSR DL250
0042: C6 01             DEC P
0044: D0 F6             BNE M2
0046: A9 00     AGAIN   LDA #0
0048: 85 04             STA T
004A: AD 00 17  POL     LDA PA
004D: 30 FB             BMI POL
004F: E6 04     PLUS1   INC T           ;KEY DOWN?
0051: AD 00 17  M3      LDA PA
0054: 10 FB             BPL M3
0056: A0 1E             LDY #$1E        ;KEY UP?
0058: A2 01     M4      LDX #1
005A: 20 9E 00          JSR TIME10
005D: AD 00 17          LDA PA
0060: 10 ED             BPL PLUS1
0062: 88                DEY
0063: 10 F3             BPL M4
                ;ANSWER COMPLETE: RESULT IN T
0065: A6 02             LDX NSAVE
0067: A4 03             LDY PSAVE
0069: 20 10 02          JSR MULTI       ;RESULT IN A
006C: C5 04             CMP T
006E: F0 0D             BEQ BRAVO
                ;WRONG ANSWER
0070: A0 10             LDY #$10
0072: 20 50 02  M5      JSR SOUND
0075: 20 90 00          JSR DL250
0078: 88                DEY
0079: D0 F7             BNE M5
007B: F0 C9             BEQ AGAIN
                ;CORRECT ANSWER
007D: A0 20     BRAVO   LDY #$20
007F: 20 50 02  M6      JSR SOUND
0082: 88                DEY
0083: D0 FA             BNE M6
0085: 00                BRK
                ;
                        .=$90
0090: 98        DL250   TYA
0091: A2 3D             LDX #$3D
0093: A0 00     DL2     LDY #0
0095: C8        DL1     INY
0096: D0 FD             BNE DL1
0098: E8                INX
0099: D0 FB             BNE DL2
009B: A8                TAY
009C: 60                RTS
                ;
                        .=$9E
009E: 86 9D     TIME10  STX D           ;DURATION IN 1/10 SEC
00A0: A9 62     T0      LDA #$62        ;98 BASE TEN
00A2: 8D 07 17          STA TIMER       ;TIMER 1024
00A5: AD 07 17  T1      LDA TIMER
00A8: 10 FB             BPL T1
00AA: C6 9D             DEC D
00AC: D0 F2             BNE T0
00AE: 60                RTS
                ;
                        .=$210
0210: 8E 00 02  MULTI   STX X
```

```
0213: 8C 01 02          STY Y
0216: A0 08             LDY #8
0218: A9 00             LDA #0
021A: 4E 00 02  ONE     LSR X
021D: 90 04             BCC TWO
021F: 18                CLC
0220: 6D 01 02          ADC Y
0223: 4A        TWO     LSR A
0224: 6E 02 02          ROR RESUL
0227: 88                DEY
0228: D0 F0             BNE ONE
022A: AD 02 02          LDA RESUL
022D: 60                RTS
                ;
                        .=$250
0250: 8D 40 02  SOUND   STA ASAVE
0253: 8E 41 02          STX XSAVE
0256: 8C 42 02          STY YSAVE
0259: A9 00             LDA #0
025B: A2 80             LDX #$80
025D: A0 00     CL2     LDY #0
025F: C8        CL1     INY
0260: D0 FD             BNE CL1
0262: 49 01             EOR #1
0264: 8D 00 17          STA PA
0267: E8                INX
0268: D0 F3             BNE CL2
026A: AD 40 02          LDA ASAVE
026D: AE 41 02          LDX XSAVE
0270: AC 42 02          LDY YSAVE
0273: 60                RTS
```

```
SYMBOL TABLE:
 N              0000        P              0001        NSAVE          0002
 PSAVE          0003        T              0004        D              009D
 X              0200        Y              0201        RESUL          0202
 ASAVE          0240        XSAVE          0241        YSAVE          0242
 PA             1700        PAD            1701        TIMER          1707
 START          0020        M1             002D        M2             003C
 AGAIN          0046        POL            004A        PLUS1          004F
 M3             0051        M4             0058        M5             0072
 BRAVO          007D        M6             007F        DL250          0090
 DL2            0093        DL1            0095        TIME10         009E
 T0             00A0        T1             00A5        MULTI          0210
 ONE            021A        TWO            0223        SOUND          0250
 CL2            025D        CL1            025F
DONE
```

APPENDICE C
LISTINGS
DES PROGRAMMES

Chapitre 4, Partie 1

```
LINE # LOC    CODE    LINE

0002 0000            ;THIS IS A SUBROUTINE WHICH ACCEPTS ASCII CHARACTERS
0003 0000            ;IN THE RANGE 2CH TO 5AH (PLUS 20H FOR SPACE) AND PLAYS
0004 0000            ;THEIR MORSE CODE EQUIVILENT ON A SPEAKER HOOKED UP TO
0005 0000            ;PB7, 6522-VIA.  IT ALSO TURNS ON AND OFF PB0, 6522-
0006 0000            ;VIA, AND WITH A SUITABLE DRIVER, THIS BIT CAN KEY A
0007 0000            ;TRANSMITTER.  A MAIN PROGRAM WILL CALL THIS SUBROUTINE
0008 0000            ;WITH THE ASCII CHARACTER IN THE ACCUMULATOR.
0009 0000            ;EXAMPLES OF THE MAIN PROGRAM WOULD BE ONE THAT
0010 0000            ;GETS INPUT FROM A TERMINAL AND SENDS MORSE CODE OUT
0011 0000            ;THROUGH THIS PROGRAM, OR A PROGRAM WHICH RANDOMIZES
0012 0000            ; A SERIES A CHARACTERS AND SENDS THEM FOR CODE PRACTICE.
0013 0000            ;THE FORMAT FOR THE MORSE CODE CAHRACTERS IN THE TABLE
0014 0000            ;IS ; MOVING FROM LEFT TO RIGHT , THE FIRST HIGH
0015 0000            ;BIT (A ONE) IS THE START BIT, AND AFTER THIS ,
0016 0000            ;EACH ONE IS A DASH, AND EACH ZERO IS A DOT.
0017 0000            SPEED=$F0
0018 0000            COUNT=$F1
0019 0000            CHAR=$F2
0020 0000                 .=$300
0021 0300: C9 20     MORSE CMP #$20        ;IF A SPACE, DO SPACE ROUTINE
0022 0302: F0 67           BEQ SPACE
0023 0304: C9 2C           CMP #$2C        ;SEE IF ASCII CODE
0024 0306: 90 4E           BCC EXIT        ;   IS LESS THEN 2CH, AND RETURN IF SO.
0025 0308: C9 5B           CMP #$5B        ;SEE IF ASCII CODE IS OVER
0026 030A: B0 4A           BCS EXIT        ;   5AH, AND RETURN IF SO
0027 030C: AA              TAX             ;PUT CODE IN INDEX REGISTER
0028 030D: BD 45 03        LDA TABLE-$2C,X ;GET MORSE CHARACTER
0029 0310: A0 08           LDY #$8         ;NUMBER OF BITS TO BE ROTATED FROM ACCUMULATOR
0030 0312: 84 F1           STY COUNT
0031 0314: 0A         STARTB ASL A
0032 0315: C6 F1           DEC COUNT
0033 0317: 90 FB           BCC STARTB      ;SHIFT A UNTIL START BIT FOUND
0034 0319: 85 F2           STA CHAR
0035 031B: A5 F2      NEXT LDA CHAR
0036 031D: 0A              ASL A           ;NOW SHIFT OUT MORSE CODE (1=DASH, 0=DOT)
0037 031E: 85 F2           STA CHAR
0038 0320: A0 01           LDY #$1         ;DOT= 1 TIME PERIOD, DEFAULT TO DOT
0039 0322: 90 02           BCC SEND        ;IF CARRY CLEAR, DOT
0040 0324: A0 03           LDY #$3         ;ELSE DASH    (3 TIME PERIODS)
```

Programme 4-1 : Morse (fig. 4-31 dans le texte)

```
0041                    ; THIS SECTION SENDS A HIGH OUTPUT FOR (Y REGISTER ) NU
0042                    ;OF TIME PERIODS, AND THEN A LOW FOR 1 TIME PERIOD.
0043  0326: A9 C0      SEND  LDA #$C0          ;SET TIMER  MODE TOFREE RUNNING MODE
0044  0328: 8D 08 A0        STA $A008
0045  032B: A9 00            LDA #$0           ; THIS VALUE,
0046  032D: 8D 06 A0        STA $A006
0047  0330: A9 04            LDA #$04          ; AND THIS VALUE DETERMINE THE TONE
0048  0332: 8D 07 A0        STA $A007         ;  OF THE OUTPUT SQUARE WAVE
0049  0335: 8D 05 A0        STA $A005         ;THIS STARTS TONE
0050  0338: A9 01            LDA #$1           ;TURN ON OUTPUT BIT-PB0
0051  033A: 8D 00 A0        STA $A000
0052  033D: 20 57 03        JSR DELAY         ;DELAYFOR ELEMENT TIME PERIOD
0053  0340: A9 00            LDA #$0
0054  0342: 8D 08 A0        STA $A008         ;TURN OFF TONE
0055  0345: 8D 00 A0        STA $A000         ;TURN OFF OUTPUT BIT -PB0)
0056  0348: A0 01            LDY #$01
0057  034A: 20 57 03        JSR DELAY         ;DELAY FOR 1 TIME PERIOD(SPACE BETWEEN ELEMENTS)
0058  034D: C6 F1            DEC COUNT         ;DECREMENT COUNT -SEE IF 8 BITS WERE ROTATED
0059  034F: D0 CA            BNE NEXT          ;  IF NOT, DO ANOTHER ELEMENT
0060  0351: A0 02      FINISH LDY #$2          ;DELAY FOR 3(TWO HERE PLUS PREVIOUS SPACE
0061  0353: 20 57 03        JSR  DELAY        ;   AT END OF LAST ELEMENT) TIME PERIODS(SPACE BET
0062  0356: 60          EXIT  RTS
0063                    ; THIS DELAYS FOR (Y REGISTER) *$FEE0*.005 SECONDS
0064  0357: 98          DELAY TYA
0065  0358: 0A                ASL A
0066  0359: 0A                ASL A
0067  035A: A8                TAY
0068  035B: A5 F0      D3    LDA SPEED
0069  035D: A2 FA      D2    LDX #$FA
0070  035F: CA          D1    DEX
0071  0360: D0 FD            BNE D1
0072  0362: 38                SEC
0073  0363: E9 01            SBC #$1
0074  0365: D0 F6            BNE D2            ;DELAY FOR 7 TIME PERIODS
0075  0367: 88                DEY               ;   (SPACE BETWEEN WORDS)
0076  0368: D0 F1            BNE D3            ;RETURN FROM MORSE PROGRAM
0077  036A: 60                RTS
0077  036B: A0 07      SPACE LDY #$7
0077  036D: 20 [illegible]    JSR DELAY
0077  0370: 60                RTS
```

Programme 4-1 : Morse (suite)

```
0077  0371: 73          TABLE  .BYTE $73,$31,$6A,$32,$3F,$2F
0077  0372: 31
0078  0373: 6A
0078  0374: 32
0078  0375: 3F
0078  0376: 2F
0078  0377: 27          .BYTE $27,$23,$21,$20,$30,$38
0078  0378: 23
0079  0379: 21
0079  037A: 20
0079  037B: 30
0079  037C: 38
0079  037D: 3C          .BYTE $3C,$3E,$01,$01,$01,$01
0079  037E: 3E
0080  037F: 01
0080  0380: 01
0080  0381: 01
0080  0382: 01
0080  0383: 01          .BYTE $01,$4C,$01,$05,$18,$1A
0080  0384: 4C
0081  0385: 01
0081  0386: 05
0081  0387: 18
0081  0388: 1A
0081  0389: 0C          .BYTE $0C,$02,$12,$0E,$10,$04
0081  038A: 02
0082  038B: 12
0082  038C: 0E
0082  038D: 10
0082  038E: 04
0082  038F: 17          .BYTE $17,$0D,$14,$07,$06,$0F
0082  0390: 0D
0083  0391: 14
0083  0392: 07
0083  0393: 06
0083  0394: 0F
0083  0395: 16          .BYTE $16,$1D,$0A,$08,$03,$09
0083  0396: 1D
0084  0397: 0A
0084  0398: 08
```

Programme 4-1 : Morse (suite)

```
0084  0399: 03
0084  039A: 09
0084  039B: 11          .BYTE $11,$0B,$19,$1B,$1C
0085  039C: 0B
0085  039D: 19
0085  039E: 1B
0085  039F: 1C

SYMBOL TABLE:
  SPEED       00F9        COUNT       00F1        CHAR        00F3
  MORSE       0300        STARTR      0314        NEXT        031B
  SEND        0326        FINISH      0351        EXIT        0356
  DELAY       0357        D3          035B        D2          035D
  D1          035F        SPACE       036B        TABLE       0371
```

Programme 4-1 : Morse (suite)

```
LINE #  LOC   CODE        LINE

                          ;FIRST LOAD A7 IN LOCATION A67E, AND 03 IN A07F
0002    0000              ;THIS IS A REAL TIME CLOCK ROUTINE WHICH MAINTAINS
0003    0000              ;THE CURRENT TIME IN THE LOCATIONS SEC (00F6), MIN
0004    0000              ;(00F5), AND HOUR (00F4) [24 HOUR TMEE]. IT IS BRANCHED TO
0005    0000              ;BY THE TIME OUT OF THE INTERRUPT TIMER, WHICH
0006    0000              ;CAUSES AN INTERRUPT AND BRANCH TO THE CLOCK
0007    0000              ;ROUTINE TWENTY TIMES PER SECOND. THE CLOCK ROUTINE
0008    0000              ;AND INTERVAL TIMER MUST BE INITIALIZED FIRST. THE
0009    0000              ;CODE 'INIT' DOES THIS, AND IT MUST BE BRANCHED TO TO
0010    0000              ;START THE CLOCK. TO INITIALIZE, PUT THE CURRENT TIME
0011    0000              ;THE CLOCK ROUTINE WILL BE STARTED IN SEC, MIN, AND
                          ;HOUR, THEN ISSUE THE COMMAND 'GO 0390 CR' AT THAT
                          ;EXACT TIME. NOTHING ELSE MUST BE DONE.
0012    0000              COUNT = $00F7          ;COUNTER FOR TWENTIETHS OF A SEC
0013    0000              SECS = $00F6           ;CURRENT TIME
0014    0000              MIN = $00F5
0015    0000              HOUR = $00F4
0016    0000              ACR = $A00B            ;TIMER MODE REGISTER
0017    0000              T1LL = $A006           ;LOW ORDER TIMER CONSTANT
0018    0000              T1HC = $A005           ;HIGH ORDER TIMER CONSTANT
0019    0000                      * = $0390
0020    0390  A9 14       INIT    LDA #$14               ;SET TO FIRST TWENTY
0021    0392  85 F7               STA COUNT              ;COUNTS
0022    0394  8D 0B A0            STA ACR                ;SET BITS 8 AND 7 LOW
                                                         ;IN ACR
0023    0397  A9 C0               LDA #$C0               ;;SET BITS 8 AND 7 HIGH IN
0024    0399  8D 0E A0            STA $A00E              ;THE INTERRUPT ENABLE
                                                         ;REGISTER (TO ENABLE
                                                         ;INTERRUPTS FROM TIMER 1)
0025    039C  A9 50               LDA #$50               ;STORE C350 IN TIMER
0026    039E  8D 06 A0            STA T1LL               ;  (DELAY CONSTANT FOR
0027    03A1  A9 C3               LDA #$C3               ;  50 MS)
0028    03A3  8D 05 A0            STA T1HC               ;THIS STARTS TIMER
0029    03A6  60                  RTS                    ;RETURN TO MONITOR
0030    03A7  08          CLOCK   PHP                    ;SAVE STATUS
0031    03A8  48                  PHA
0032    03A9  F8                  SED
0033    03AA  A9 50               LDA #$50               ;STORE C350 IN TIMER
0034    03AC  8D 06 A0            STA T1LL               ;  (DELAY CONSTANT FOR
0035    03AF  A9 C3               LDA #$C3               ;  50 MS)
0036    03B1  8D 05 A0            STA T1HC               ;THIS STARTS TIMER
0037    03B4  C6 F7               DEC COUNT              ;DECREMENT COUNT OF
                                                         ;TWENTY
0038    03B6  D0 31               BNE EXIT               ;EXIT IF WE HAVE NOT
                                                         ;COUNTED TO TWENTY YET
0039    03B8  A9 14               LDA #$14               ;ELSE RESTORE COUNT—
0040    03BA  85 F7               STA COUNT              ;A FULL SECOND HAS PASSED
0041    03BC  A9 01               LDA #$01
0042    03BE  18                  CLC
0043    03BF  65 F6               ADC SECS               ;ADD 1 TO SEC
0044    03C1  85 F6               STA SECS
0045    03C3  C9 60               CMP #$60               ;SEE IF 60 SECONDS
0046    03C5  D0 22               BNE EXIT               ;IF NOT, EXIT
0047    03C7  A9 00               LDA #$00               ;ELSE RESET SECONDS TO 0
0048    03C9  85 F6               STA SECS
0049    03CB  A9 01               LDA #$01
0050    03CD  18                  CLC
```

Programme 4-2 : Horloge 24 heures (fig. 4-37 dans le texte)

```
0051  03CE  65 F5          ADC MIN          ;AND ADD 1 TO MINUTES
0052  03D0  85 F5          STA MIN
0053  03D2  C9 60          CMP #$60         ;SEE IF 60 MINUTES
0054  03D4  D0 13          BNE EXIT         ;IF NOT, EXIT
0055  03D6  A9 00          LDA #$00
0056  03D8  85 F5          STA MIN          ;ELSE RESET MINUTES TO 0
0057  03DA  A9 01          LDA #$01
0058  03DC  18             CLC
0059  03DD  65 F4          ADC HOUR         ;AND ADD 1 TO HOUR
0060  03DF  85 F4          STA HOUR
0061  03E1  C9 24          CMP #$24         ;SEE IF 24 HOURS
0062  03E3  D0 04          BNE EXIT         ;IF NOT, EXIT
0063  03E5  A9 00          LDA #$00
0064  03E7  85 F4          STA HOUR         ;ELSE RESET HOUR TO 0
0065  03E9  68       EXIT  PLA              ;RESTORE STATUS
0066  03EA  28             PLP
0067  03EB  40             RTI

ERRORS = 0000 <0000>

SYMBOL TABLE

SYMBOL    VALUE

ACR       A00B    CLOCK     03A7    COUNT     00F7    EXIT      03E9
HOUR      00F4    INIT      0390    MIN       00F5    PLS       03EA
SECS      00F6    T1HC      A005    T1LL      A006

END OF ASSEMBLY
```

Programme 4-2 : Horloge 24 heures (suite)

```
LINE #  LOC   CODE        LINE
0002    0000              ;THIS IS A SIMPLE HOME CONTROL ROUTINE WHICH RUNS
0003    0000              ;THROUGH A LOOP. EACH TIME THROUGH IT DISPLAYS THE
0004    0000              ;CURRENT TIME AND BRANCHES TO A NUMBER OF USER
                          SUBROUTINES
0005    0000              ;WHICH SERVICE DEVICES.
0006    0000              ;EXAMPLES:
0007    0000              ;1) A SUBROUTINE COULD CHECK THE CURRENT TIME AND
0008    0000              ;    TURN ON A LIGHT IF THE TIME WERE RIGHT.
0009    0000              ;2) A SUBROUTINE COULD MONITOR THE STATUS OF AN
0010    0000              ;    ALARM SYSTEM AND TAKE APPROPRIATE ACTION IF AN
0011    0000              ;    INTRUDER WERE DETECTED.
0012    0000              DDRB = $AC02
0013    0000              IORB = $AC00
0014    0000              HOUR = $00F4
0015    0000              MIN = $00F5
0016    0000              OUTBYT = $82FA
0017    0000              SCAND = $8906
0018    0000                       * = $0200
0019    0200  D8          CONTRL   CLD                    ;
0020    0201  A9 OF                LDA #$0F             ;SET DATA DIRECTION
0021    0203  8D 02 AC             STA DDRB             ;REGISTER TO OUTPUT FOR
                                                        RELAYS
0022    0206  A9 00                LDA #$00
0023    0208  8D 00 AC             STA IORB             ;TURN OFF RELAYS
0024    020B  A5F4       LOOP      LDA HOUR             ;THIS IS THE MAIN CONTROL
                                                        LOOP
0025    020D  20 FA 82             JSR OUTBYT           ;OUTPUT CURRENT HOUR TO
                                                        DISPLAY
0026    0210  A5 F5                LDA MIN
0027    0212  20 FA 82             JSR OUTBYT           ;OUTPUT CURRENT MINUTE
                                                        TO DISPLAY
0028    0215  20 06 89             JSR SCAND            ;REFRESH (LIGHT) DISPLAY
                                                        WITH TIME
0029    0218  EA                   .BYTE $EA,$EA,$EA
0029    0219  EA
0029    021A  EA
0030    021B  EA                   .BYTE $EA,$EA,$EA
0030    021C  EA
0030    021D  EA
0031    021E  EA                   .BYTE $EA,$EA,$EA
0031    021F  EA
0031    0220  EA
0032    0221  EA                   .BYTE $EA,$EA,$EA
0032    0222  EA
0032    0223  EA
0033    0224  EA                   .BYTE $EA,$EA,$EA
0033    0225  EA                                        ;THE USER CAN PLACE
                                                        JUMPS TO
0033    0226  EA                                        ;SUBROUTINES HERE TO SER-
                                                        VICE DEVICES
0034    0227  EA                   .BYTE $EA,$EA,$EA
0034    0228  EA
0034    0229  EA
0035    022A  EA                   .BYTE $EA,$EA,$EA
0035    022B  EA
0035    022C  EA
0036    022D  EA                   .BYTE $EA,$EA,$EA
0036    022E  EA
```

Programme 4-3 : Contrôle domestique (fig. 4-38 dans le texte)

```
0036    022F    EA
0037    0230    EA                      .BYTE $EA,$EA,$EA
0037    0231    EA
0037    0232    EA
0038    0233    EA                      .BYTE $EA,$EA,$EA

0038    0234    EA
0038    0235    EA
0039    0236    4C 0B 02                JMP LOOP
0040    0239

ERRORS = 0000<0000>

SYMBOL TABLE

SYMBOL    VALUE

CONTRL    0200      DDRB    AC02      HOUR    00F4      IORB    AC00
LOOP      020B      MIN     00F5      OUTBYT  82FA      SCAND   8906

END OF ASSEMBLY
```

Programme 4-3 : Contrôle domestique (suite)

```
LINE #  LOC   CODE        LINE

0002    0000              ;THIS IS A PROGRAM WHICH DIALS PRE STORED
0003    0000              ;TELEPHONE NUMBERS. IT PRODUCES A TWO TONE OUTPUT
0004    0000              ;THROUGH A SPEAKER HOOKED UP IN CONFIGURATION 2
0005    0000              ;(TWO TONES—SEE SPEAKER). THESE TONES WILL ACTIVATE
0006    0000              ;A STANDARD TOUCH TONE PHONE WHEN THE SPEAKER IS
0007    0000              ;PLACED DIRECTLY OVER THE MOUTH PIECE OF THE TELE-
0008    0000              ;PHONE. TO USE THE PROGRAM, PLACE THE PHONE
0009    0000              ;NUMBER(S) ANYWHERE IN MEMORY, ONE DIGIT PER BYTE,
0010    0000              ;AND ENDING WITH OF (HEX). FOR EXAMPLE, THE NUMBER
0011    0000              ;555-1212 WOULD BE 05 05 05 01 02 01 02 0F (ALL HEX) IN
0012    0000              ;MEMORY. THEN PLACE THE ADDRESS OF THE NUMBER,
0013    0000              ;LOW BYTE FIRST, IN THE LOCATIONS 00C0 AND 00C1.
0014    0000              ;THEN EITHER GO TO THIS ROUTINE FROM THE MONITOR
                          ;OR JSR TO IT FROM ANOTHER PROGRAM.
0015    0000              NUMPTR = $00C0      ;THIS POINTS TO THE ADDRESS OF
                                              ;THE TELEPHONE NUMBER
0016    0000              ONDEL = $40         ;THIS IS THE DELAY CONSTANT FOR
                                              ;THE TIME WHEN THE
0017    0000              OFFDEL = $20        ;DELAY CONSTANT FOR THE TIME
                                              ;WHEN THE TONES ARE 0
0018    0000              DELCON = $FF        ;GENERAL PURPOSE DELAY
                                              ;CONSTANT
0019    0000              ACR1 = $A00B        ;THESE ARE THE TIMER MODE
                                              ;REGISTERS (TIMER 1)
0020    0000              ACR2 = $AC0B        ;(TIMER 2)
0021    0000              T1CH = $A005        ;THIS IS THE TIMER 1 COUNTER
                                              ;(HIGH BYTE)
0022    0000              T1LH = $A007        ;TIMER 1 LATCH (HIGH BYTE)
0023    0000              T1LL = $A004        ;  (LOW BYTE)
0024    0000              T2CH = $AC05        ;SAME AS TIMER 1 — FOR TIMER 2
0025    0000              T2LH = $AC07
0026    0000              T2LL = $AC04
0027    0000                    * = $0300
0028    0300  A0 00       PHONE LDY #$00              ;INDEX FOR DIGITS OF
                                                      ;PHONE NUMBER
0029    0302  B1 C0       DIGIT LDA (NUMPTR)Y         ;GET DIGIT
0030    0304  C8                INY
0031    0305  C9 0F              CMP #$0F             ;SEE IF END OF PHONE
                                                      ;NUMBER
0032    0307  D0 01              BNE NOEND
0033    0309  60                 RTS                  ;RETURN IS SO (TO
                                                      ;MONITOR OR CALLING
                                                      ;PROGRAM)
0034    030A  0A EA EA    NOEND  ASL A                ;MULTIPLY NUMBER BY
                                                      ;FOUR TO INDEX TABLE
0035    030D  0A EA EA           ASL A                ;  (EACH TABLE ENTRY IS
                                                      ;  4 BYTES)
0036    0310  AA                 TAX                  ;X = INDEX FOR TABLE
0037    0311  A9 C0              LDA #$C0
0038    0313  8D 0B A0           STA ACR1             ;SET TIMER MODE TO FREE
                                                      ;RUNNING ON BOTH TIMERS
0039    0316  8D 0B AC           STA ACR2
0040    0319  BD 5D 03           LDA TABLE,X          ;GET LOW ORDER, FIRST
                                                      ;TONE
0041    031C  8D 04 A0           STA T1LL             ;STORE IN TIMER 1
0042    031F  E8                 INX
0043    0320  BD 5D 03           LDA TABLE,X          ;GET HIGH ORDER, FIRST
                                                      ;TONE
```

Programme 4-4 : Composeur de numéros de téléphone (fig. 4-41 dans le texte)

```
0044  0323  8D 07 A0          STA T1LH          ;STORE TIMER 1
0045  0326  8D 05 A0          STA T1CH          ;THIS STARTS TIMER 1
                                                ;GOING
0046  0329  E8                INX
0047  032A  BD 5D 03          LDA TABLE,X       ;GET LOW ORDER, SECOND
                                                ;TONE
0048  032D  8D 04 AC          STA T2LL          ;STORE IN TIMER 2
0049  0330  E8                INX
0050  0331  BD 5D 03          LDA TABLE,X       ;GET HIGH ORDER, SECOND
                                                ;TONE
0051  0334  8D 07 AC          STA T2LH          ;STORE IN TIMER 2
0052  0337  8D 05 AC          STA T2CH          ;THIS STARTS TIMER 2
                                                ;GOING
0053  033A  A2 40             LDX #ONDEL        ;GET TONES-ON DELAY
                                                ;CONSTANT
0054  033C  20 55 03    ON    JSR DELAY         ;DELAY WHILE TONE IS ON
0055  033F  CA                DEX
0056  0340  DO FA             BNE ON
0057  0342  A9 00             LDA #$00
0058  0344  8D 0B A0          STA ACR1          ;TURN BOTH TIMERS OFF
0059  0347  8D 0B AC          STA ACR2
0060  034A  A2 20             LDX #OFFDEL       ;GET TONES-OFF DELAY
                                                ;CONSTANT
0061  034C  20 55 03    OFF   JSR DELAY         ;DELAY WHILE TONE IS OFF
0062  034F  CA                DEX
0063  0350  D0 FA             BNE OFF
0064  0352  4C 02 03          JMP DIGIT         ;GO BACK FOR NEXT DIGIT
                                                ;OF PHONE NUMBER
0065  0355                   ;
0066  0355                   ;THIS IS A SIMPLE DELAY ROUTINE FOR THE TONE ON AND
                             ;OFF PERI
0067  0355                   ;
0068  0355  A9 FF       DELAY LDA #DELCON       ;GET DELAY CONSTANT
0069  0357  38           WAIT  SEC               ;DELAY FOR THAT LONG
0070  0358  E9 01             SBC #$01
0071  035A  D0 FB             BNE WAIT
0072  035C  60                RTS
0073  035D                   ;
0074  035D                   ;THIS IS A TABLE OF THE CONSTANTS FOR THE TONE
0075  035D                   ;FREQUENCIES FOR EACH TELEPHONE DIGIT. THE
0076  035D                   ;CONSTANTS ARE TWO BYTES LONG, LOW BYTE FIRST.
0077  035D                   ;
0078  035D  13       TABLE  .BYTE $13,$02,$76,$01   ;TWO TONES FOR '0'
0078  035E  02
0078  035F  76
0078  0360  01
0079  0361  CD               .BYTE $CD,$02,$9E,$01  ;TWO TONES FOR '1'
0079  0362  02
0079  0363  9E
0079  0364  01
0080  0365  CD               .BYTE $CD,$02,$76,$01  ;   '2'
0080  0366  02
0080  0367  76
0080  0368  01
0081  0369  CD               .BYTE $CD,$02,$53,$01  ;   '3'
0081  036A  02
0081  036B  53
0081  036C  01
0082  036D  89               .BYTE $89,$02,$9E,$01  ;   '4'
0082  036E  02
```

Programme 4-4 : Composeur de numéros de téléphone (suite)

```
0082    036F    9E
0082    0370    01
0083    0371    89                      .BYTE $89,$02,$76,$01   ;   '5'
0083    0372    02
0083    0373    76
0083    0374    01
0084    0375    89                      .BYTE $89,$02,$53,$01   ;   '6'
0084    0376    02
0084    0377    53
0084    0378    01
0085    0379    4B                      .BYTE $4B,$02,$9E,$01   ;   '7'
0085    037A    02
0085    037B    9E
0085    037C    01
0086    037D    4B                      .BYTE $4B.$02,$76,$01   ;   '8'
0086    037E    02
0086    037E    76
0086    0380    01
0087    0381    4B                      .BYTE $4B,$02,$53,$01   ;   '9'
0087    0382    02
0087    0383    53
0087    0384    01
0088    0385                            .END

ERRORS = 0000 <0000>

SYMBOL TABLE

SYMBOL    VALUE

ACR1      A00B      ACR2      AC0B      DELAY     0355      DELCON    00FF
DIGIT     0302      NOEND     030A      NUMPTR    00C0      OFF       034C
OFFDEL    0020      ON        033C      ONDEL     0040      PHONE     0300
T1CH      A005      T1LH      A007      T1LL      A004      T2CH      AC05
T2LH      AC07      T2LL      AC04      TABLE     035D      WAIT      0357

END OF ASSEMBLY
```

Programme 4-4 : Composeur de numéros de téléphone (suite)

APPENDICE D

TABLE DE CONVERSION HEXADÉCIMALE

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

5		4		3		2		1		0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393.216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

APPENDICE E

TABLE DE CONVERSION ASCII

HEX	BITS	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SPACE	0	@	P	–	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB		7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
	1111	SI	US	/	?	O	←	o	DEL

THE ASCII SYMBOLS

NUL —Null
SOH —Start of Heading
STX —Start of Text
ETX —End of Text
EOT —End of Transmission
ENQ —Enquiry
ACK —Acknowledge
BEL —Bell
BS —Backspace
HT —Horizontal Tabulation
LF —Line Feed
VT —Vertical Tabulation
FF —Form Feed
CR —Carriage Return
SO —Shift Out
SI —Shift In

DLE —Data Link Escape
DC —Device Control
NAK —Negative Acknowledge
SYN —Synchronous Idle
ETB —End of Transmission Block
CAN —Cancel
EM —End of Medium
SUB —Substitute
ESC —Escape
FS —File Separator
GS —Group Separator
RS —Record Separator
US —Unit Separator
SP —Space (Blank)
DEL —Delete

APPENDICE F

INSTRUCTIONS DU 6502

(ORDRE ALPHABÉTIQUE)

ADC	Addition avec retenue
AND	ET logique
ASL	Décalage arithmétique gauche
BCC	Branchement si C = 0
BCS	Branchement si C = 1
BEQ	Branchement si résultat = 0
BIT	Test de bits
BMI	Branchement si négatif
BNE	Branchement si non égal à 0
BPL	Branchement si positif ou nul
BRK	Interruption simulée
BVC	Branchement si non débordement
BVS	Branchement si débordement
CLC	Annuler retenue
CLD	Annuler le mode décimal
CLI	Annuler inhibition des interruptions
CLV	Annuler débordement
CMP	Comparer à l'accumulateur
CPX	Comparer à X
CPY	Comparer à Y
DEC	Décrémenter mémoire
DEX	Décrémenter X
DEY	Décrémenter Y
EOR	OU exclusif
INC	Incrémenter mémoire
INX	Incrémenter X
INY	Incrémenter Y
JMP	Saut inconditionnel
JSR	Appel de sous-programme
LDA	Charger accumulateur
LDX	Charger X
LDY	Charger Y
LSR	Décalage logique à droite
NOP	Pas d'opération
ORA	OU logique
PHA	Empiler A
PHP	Empiler P
PLA	Dépiler A
PLP	Dépiler P
ROL	Rotation à gauche
ROR	Rotation à droite
RTI	Retour d'interruption
RTS	Retour de sous-programme
SBC	Soustraction avec retenue
SEC	Mettre retenue à 1
SED	Mettre en mode décimal
SEI	Inhiber les interruptions
STA	Ranger l'accumulateur
STX	Ranger X
STY	Ranger Y
TAX	Transférer A dans X
TAY	Transférer A dans Y
TSX	Transférer S dans X
TXA	Transférer X dans A
TXS	Transférer X dans S
TYA	Transférer Y dans A

APPENDICE G

INSTRUCTIONS DU 6502 : HEXA ET TEMPS

		IMPLIED			ACCUM.			ABSOLUTE			ZERO PAGE			IMMEDIATE			ABS. X		
MNEMONIC		OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#
ADC	(1)							6D	4	3	65	3	2	69	2	2	7D	4	3
AND	(1)							2D	4	3	25	3	2	29	2	2	3D	4	3
ASL					0A	2	1	0E	6	3	06	5	2				1E	7	3
BCC	(2)																		
BCS	(2)																		
BEQ	(2)																		
BIT								2C	4	3	24	3	2						
BMI	(2)																		
BNE	(2)																		
BPL	(2)																		
BRK		00	7	1															
BVC	(2)																		
BVS	(2)																		
CLC		18	2	1															
CLD		D8	2	1															
CLI		58	2	1															
CLV		B8	2	1															
CMP								CD	4	3	C5	3	2	C9	2	2	DD	4	3
CPX								EC	4	3	E4	3	2	E0	2	2			
CPY								CC	4	3	C4	3	2	C0	2	2			
DEC								CE	6	3	C6	5	2				DE	7	3
DEX		CA	2	1															
DEY		88	2	1															
EOR	(1)							4D	4	3	45	3	2	49	2	2	5D	4	3
INC								EE	6	3	E6	5	2				FE	7	3
INX		E8	2	1															
INY		C8	2	1															
JMP								4C	3	3									
JSR								20	6	3									
LDA	(1)							AD	4	3	A5	3	2	A9	2	2	BD	4	3
LDX	(1)							AE	4	3	A6	3	2	A2	2	2			
LDY	(1)							AC	4	3	A4	3	2	A0	2	2	BC	4	3
LSR					4A	2	1	4E	6	3	46	5	2				5E	7	3
NOP		EA	2	1															
ORA								0D	4	3	05	3	2	09	2	2	1D	4	3
PHA		48	3	1															
PHP		08	3	1															
PLA		68	4	1															
PLP		28	4	1															
ROL					2A	2	1	2E	6	3	26	5	2				3E	7	3
ROR					6A	2	1	6E	6	3	66	5	2				7E	7	3
RTI		40	6	1															
RTS		60	6	1															
SBC	(1)							ED	4	3	E5	3	2	E9	2	2	FD	4	3
SEC		38	2	1															
SED		F8	2	1															
SEI		78	2	1															
STA								8D	4	3	85	2					9D	5	3
STX								8E	4	3	86	2							
STY								8C	4	3	84	2							
TAX		AA	2	1															
TAY		A8	2	1															
TSX		BA	2	1															
TXA		8A	2	1															
TXS		9A	2	1															
TYA		98	2	1															

(1) Add 1 to n if crossing page boundary

(2) Add 2 to n if branch within page
Add 3 to n if branch to another page

ABS. Y			(IND. X)			(IND)Y			Z. PAGE, X			RELATIVE			INDIRECT			Z. PAGE, Y			PROCESSOR STATUS CODES
OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	N V B D I Z C
79 39	4 4	3 3	61 21	6 6	2 2	71 31	5 5	2 2	75 35 16	4 4 6	2 2 2	 90 B0	 2 2	 2 2							● ● ● ● ● ●
												FO 30 DO 10	2 2 2 2	2 2 2 2							M₇ M₆ ●
												50 70	2 2	2 2							1 1 0 0
D9	4	3	C1	6	2	D1	5	2	D5	4	2										0 0 ● ● ● ● ● ● ● ● ●
59	4	3	41	6	2	51	5	2	D6 55 F6	ø6 4 6	2 2 2										● ● ● ● ● ● ●
B9	4	3	A1	6	2	B1	5	2	B5	4	2				6C	5	3				● ● ● ●
BE 19	4	3 3	01	6	2	11	5	2	B4 56 15	4 6 4	2 2 2							B6	4	2	● ● ● 0 ● ● ● ●
									 36	 6	 2										● ● ● ● ● ● ● ● ● ● ●
F9	4	3	E1	6	2	F1	5	2	76 F5	6 4	2 2										● ● ● ● ● ● ● ● ● ● ● ● 1 1
99	5	3	B1	6	2	91	6	2	95 94	4 4	2 2							96	4	2	1 ● ●
																					● ● ● ●

INDEX DES PROGRAMMES

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks,*
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks,*
636 pages, Réf. 340

TECHNIQUES D'INTERFACE aux microprocesseurs *par Austin Lesea et Rodnay Zaks,*
450 pages, Réf. 339

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par Joël Poncet,*
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux,*
140 pages, Réf. 369

LA SOLUTION RS-232 *par Joe Campbell,*
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par Pierrick Bourgault,*
198 pages, Réf. 0119

BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux,*
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par Jean-Pierre Lamoitier,*
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui,*
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller,*
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE *par Richard Mateosian,*
352 pages, Réf. 429

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
204 pages, Réf. 247

FICHIERS EN BASIC *par Alan Simpson,*
256 pages, Réf. 0102

PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux,*
496 pages, Réf. 330

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 360

MICRO-ORDINATEURS

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par Norbert Rimoux,*
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II, Tomes 1 et 2 *par Léopold Laurent,*
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
par Douglas Hergert et Joseph T. Kalash,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert,*
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar,
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
par Goulven Habasque,
240 pages, Réf. 0099

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bunn,*
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar,*
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut,*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost,*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kascmer,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Ravis,*
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Dejonghe et Caroline Earhart,
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
par Gary Lippman,
216 pages, Réf. 0100

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut,*
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella,
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier,*
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
par Joan Lasselle et Carol Ramsey,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par Nelson Ford,*
320 pages, Réf. 357

GUIDE DU PC DOS *par Richard A. King,*
240 pages, Réf. 0013

LASER

LASER JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre,*
272 pages, Réf. 0118

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw,*
192 pages, Réf. 0132

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee,
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella,
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly,*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
par Jean-Pierre Lhoir,
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 374

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent,
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut,
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Orson Kellog,
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
par Stanley R. Trost,
160 pages, Réf. 413

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann,
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 358

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 274

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

'ROGRAMMATION DU 6800
ar Daniel-Jean David et Rodnay Zaks,
74 pages, Réf. 327

ROGRAMMATION DU 6809
ar Rodnay Zaks et William Labiak,
92 pages, Réf. 0139

ROGRAMMATION DU 8086/8088
ar James W. Coffron,
04 pages, Réf. 0016

IISE EN OEUVRE DU 68000 *par C. Vieillefond,*
52 pages, Réf. 0133

SSEMBLEUR DU 8086/8088
ar François Retoreau,
16 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

UIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
54 pages, Réf. 336

P/M APPROFONDI *par Alan R. Miller,*
80 pages, Réf. 334

NTRODUCTION AU p-SYSTEM UCSD
ar Charles W. Grant et Jon Butah,
08 pages, Réf. 365

UIDE DE MS-DOS *par Richard A. King,*
60 pages, Réf. 0117

APPLICATIONS ET LOGICIELS

NTRODUCTION AU TRAITEMENT DE TEXTE
ar Hal Glatzer,
28 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par Julie Anne Arca,*
320 pages, Réf. 0005

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson,*
280 pages, Réf. 0064

DE VISICALC A VISI ON *par Jacques Bourdeu,*
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
par D. Helle et G. Boussand,
304 pages, Réf. 0079

dBASE II APPLICATIONS *par Christophe Stehly,*
248 pages, Réf. 416

INTRODUCTION A LOTUS 1-2-3
par Chris Gilbert et Laurie Williams,
272 pages, Réf. 0106

LOGISTAT, ANALYSE STATISTIQUE DES DONNÉES
par Fredj Tekaia et Michele Bidel,
192 pages, Réf. 0132

La plupart de ces ouvrages existent en version anglaise.

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 203.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 626441
Telex : 08588163

Paris • Berkeley • Düsseldorf

Achevé d'imprimer le 5 septembre 1985 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 3e trimestre 1985 - Imprimeur n° 1853

Ce livre présente des techniques d'applications pratiques pouvant être mises en œuvre sur le microprocesseur 6502. Grâce à lui, vous pourrez construire un système d'alarme complet pour une habitation (comprenant un détecteur d'incendie), un piano électronique, un régulateur de vitesse de moteur, une horloge 24 heures, un système de commande de feux de carrefour simulés et un générateur de Morse. Vous concevrez une boucle de régulation industrielle de température (comprenant un convertisseur analogique-digital) et vous connecterez des périphériques simples : lecteur de ruban perforé ou micro-imprimante. C'est véritablement le livre des «entrées-sorties» du 6502. Il comporte plus de 50 exercices pour vous permettre de vérifier vos connaissances à chaque étape.

L'AUTEUR

a enseigné la programmation et les micro-processeurs à plus de 4 000 personnes, dans le monde entier. Titulaire d'un doctorat en informatique de l'Université de Californie (Berkeley), il a développé un APL micro-programmé et a travaillé, à Silicon Valley, sur des systèmes industriels à microprocesseurs, dès leur apparition. Ce livre, comme les autres de cette série, est le résultat de son expérience technique et pédagogique.
R. ZAKS est l'auteur de plusieurs best-sellers sur les micro-ordinateurs, actuellement traduits en dix langues.

0032 0985 128 F
9 782736 100315

SYBEX

RODNAY ZAKS APPLICATIONS DU 6502