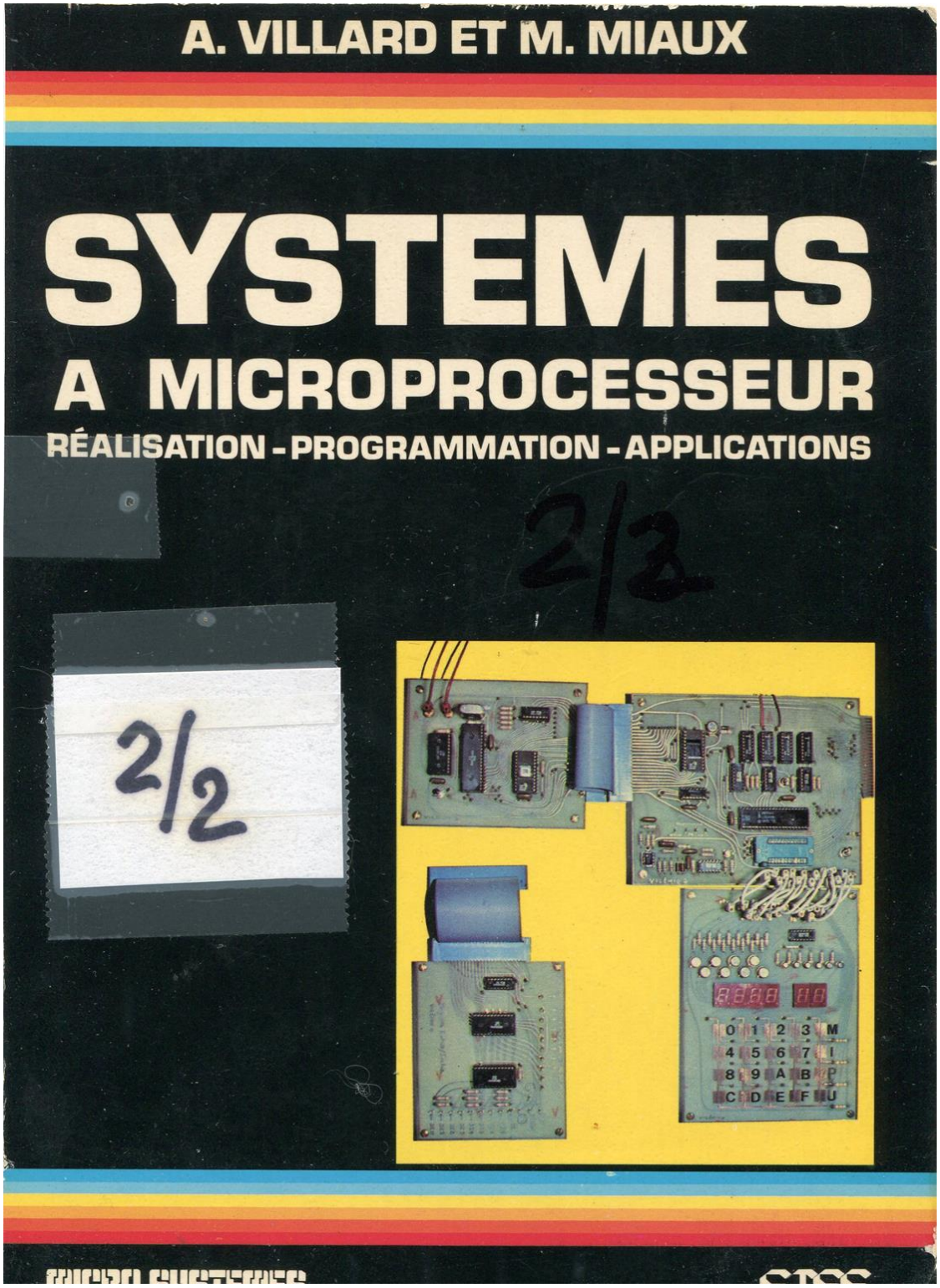
Tome no 2 Microprocesseur 1802 pas-à-pas

Format 150x120 to A4 – 300 dpi
Scanner EPSON V33 – PRO
Scanné par Alain Pilloud
Le 18-19/08/2021



André VILLARD Michel MIAUX

Anciens élèves de l'Ecole Normale Supérieure
de l'Enseignement Technique
Professeurs Agrégés au LT Dorian à Paris

Systèmes à microprocesseur

réalisation - programmation - applications

Diffusion :
ÉDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue, 75940 PARIS CEDEX 19

SOMMAIRE

Première partie

Conception et réalisation pratique du système VILÉMIO

Deuxième partie

Mode d'utilisation et conception logicielle du système VILÉMIO Ses compléments : interface cassette et programmateur d'EPROM

Chapitre 5. Fonction et procédure d'utilisation du moniteur

Chapitre 6. L'affichage. La scrutation et l'encodage du clavier

Chapitre 9. Programmateur d'EPROM

Chapitre 10. Enregistrement et lecture sur cassette d'une série d'octets

Chapitre 11. Compléments au programme moniteur

Troisième partie

Quelques idées pour vos projets à microprocesseur

Chapitre 13. La commande d'un moteur pas à pas

Chapitre 14. La mesure des grandeurs physiques

Première partie

Conception et réalisation pratique du système VILÉMIO

Chapitre 1

PRÉSENTATION DE L'OUVRAGE ET DU MICROPROCESSEUR UTILISÉ

I. Premières idées

Dans notre précédent ouvrage « Un Microprocesseur pas à pas » (1), nous nous sommes placés au point de vue de l'électronicien désireux d'insérer le microprocesseur, considéré alors comme un composant électronique comme les autres, dans un montage correspondant à une application donnée. Pour ce faire, nous avons tenté de démythifier le microprocesseur en permettant son libre accès par l'intermédiaire d'une maquette facile à réaliser, de structure très simple, et immédiatement opérationnelle. En nous plaçant ainsi sur un terrain résolument pédagogique, nous pensons avoir contribué à la « popularisation » du microprocesseur. Peut-être avons-nous aussi réussi à jeter un pont entre l'électronique et l'informatique, et ainsi montré que ces disciplines ne peuvent constituer des domaines fermés, mais au contraire ont intérêt à fusionner pour former un amalgame dont l'étiquette est à définir.

Abordons maintenant ce nouvel ouvrage « Systèmes à microprocesseur ».

On peut considérer a priori qu'il existe deux *genres* différents de systèmes à microprocesseurs :

1. Les systèmes matérialisant une fonction répondant à une application bien déterminée. Il s'agit de montages dont la fonction ne peut être modifiée. Par exemple, on peut utiliser un microprocesseur pour réaliser une horloge donnant l'heure, la date, et permettant l'enclenchement et l'arrêt de divers appareils à commander, ceci à des époques programmables. D'une façon générale, tous les montages que l'électronicien avait l'habitude de concevoir en utilisant des composants « classiques », comme les ther-

(1) Villard et Miaux, « Un Microprocesseur pas à pas », chez le même éditeur.

momètres, les tachymètres, les horloges, etc., peuvent être réalisés avec des microprocesseurs. Les machines à jouer aux échecs, à laver le linge, celles qui sont destinées au contrôle des locaux ou de la cuisson des poulets, sont souvent, ou en tout cas seront, des systèmes à microprocesseurs. La propriété de ces montages est de remplir une fonction bien définie et immuable (une machine à jouer aux échecs ne peut contrôler la cuisson d'un poulet !). La structure logicielle de ces systèmes est définie par le contenu d'une mémoire *morte* (ROM ou EPROM) constituant le programme de « gestion » de l'ensemble.

2. Les systèmes appelés couramment « micro-ordinateurs » (ou, au choix : mini-ordinateurs, ordinateurs) pour lesquels on ne peut parler d'applications spécifiques puisque celles-ci sont programmables à volonté. En effet, au prix de la réalisation des interfaces appropriés, un même micro-ordinateur peut jouer aux échecs et contrôler la cuisson d'un poulet. Ce même micro-ordinateur permet la simulation d'une horloge, d'un thermomètre, d'un tachymètre, etc. Il permet aussi la mise en œuvre de montages pour lesquels l'accent est mis sur le « traitement des données », comme le tri, le calcul, etc. Un micro-ordinateur est donc un système à microprocesseur contenant une mémoire *vive* (RAM) destinée à recevoir temporairement le programme de « gestion » de l'application envisagée.

Lorsque nous avons entrepris la rédaction de cet ouvrage, notre première idée était de proposer au lecteur tous les moyens propres à lui permettre de réaliser une application à microprocesseur obéissant à une fonction bien déterminée. Nous avons donc imaginé un système, dit « système VILÉMIO », modulaire, répondant à ces spécifications. Lorsque notre travail fut terminé, nous nous sommes aperçus presque avec surprise que VILÉMIO présentait toutes les caractéristiques d'un micro-ordinateur, et qu'à ce titre il permettait *aussi* la mise en œuvre de toutes les applications qui s'y rattachaient, et auxquelles nous n'avions pas pensé de prime abord.

II. Nos propositions

II. 1. La maquette processeur

Supposez que vous désiriez réaliser un montage électronique à microprocesseur obéissant à une fonction déterminée, peu importe laquelle pour le moment. Ce problème revêt deux aspects :

a) l'aspect *matériel.* Il s'agit de déterminer quels sont les composants qui vont intervenir dans votre montage, quelles sont leurs interconnexions ;

b) l'aspect *logiciel.* Il s'agit de déterminer le contenu de la mémoire de programme pour que le but que vous vous êtes fixé se réalise exactement selon vos prévisions.

Il ne faut pas croire que ces deux aspects soient indépendants. En effet, pour déterminer la structure matérielle de l'ensemble à réaliser, il faut (et on

s'en rendra compte au cours de la progression) avoir une bonne idée, sinon une idée intime, des possibilités logicielles du microprocesseur utilisé. Inversement, la structure du programme dépend fortement de la structure matérielle adoptée. D'une façon générale, on peut considérer un montage à microprocesseur comme une balance dont l'un des plateaux supporte le « matériel », et l'autre le « logiciel » : si le « matériel » est lourd, le logiciel (c'est-à-dire la structure et la complexité du programme inscrit dans la mémoire) est simple et rapide à mettre au point ; c'est cependant un mauvais équilibre puisqu'à la limite on perd le bénéfice apporté par l'utilisation de la logique programmée, bénéfice qui consiste précisément dans la limitation du nombre de composants et dans la souplesse de mise au point finale qui en résulte. Si le « matériel » est réduit au strict minimum, le « logiciel » risque d'être lourd, et donc coûter de nombreuses heures (sinon des semaines) de recherches ; il risque aussi de demander beaucoup d'espace mémoire, et de nuire à la rapidité de réaction demandée à tout système électronique qui se respecte. Dans ce cas, la balance n'est pas mieux équilibrée. C'est une certaine expérience des systèmes à microprocesseurs qui permet de déterminer le bon « poids » à donner au matériel et au logiciel. Chaque cas est particulier, chaque application nécessite une réflexion propre, et il ne serait ni judicieux ni élégant de demander à une structure matérielle type et hypothétique, rigide, d'accomplir des fonctions très différentes.

Cependant, revenons à notre problème (que nous supposons être aussi le vôtre) consistant à réaliser un montage obéissant à une fonction déterminée. Bien que cette dernière soit encore inconnue, ou susceptible de changement, il faut constater que dans tous les cas, la structure matérielle présente une partie qui est toujours la même : il s'agit de *l'ensemble microprocesseur-mémoire associée.* Quelle que soit l'application considérée, on retrouvera cet ensemble qu'on peut donc construire une bonne fois pour toutes. Nous appelons cet ensemble : *« maquette processeur ».* Puisque nous désirons construire un montage autonome, qui marche, il faut bien comprendre que la mémoire associée au microprocesseur comporte une partie mémoire morte, celle-ci contenant le programme. L'autre partie de la mémoire associée au microprocesseur est une mémoire vive destinée à contenir temporairement des données stockées par le microprocesseur lui-même ou par l'utilisateur. Pour la maquette processeur, nous avons décidé de prendre comme taille de la mémoire morte (EPROM, c'est-à-dire mémoire morte programmable et effaçable) la capacité de 2 K octets (2048 octets) ce qui est amplement suffisant pour un très grand nombre d'applications. La capacité de la mémoire vive (RAM) est de 128 octets. Le cas échéant, la taille de la mémoire associée peut être augmentée.

II. 2. Les maquettes application

La figure I. 1 montre à quoi sert la maquette processeur. Celle-ci doit naturellement être couplée à une maquette dite « maquette application »,

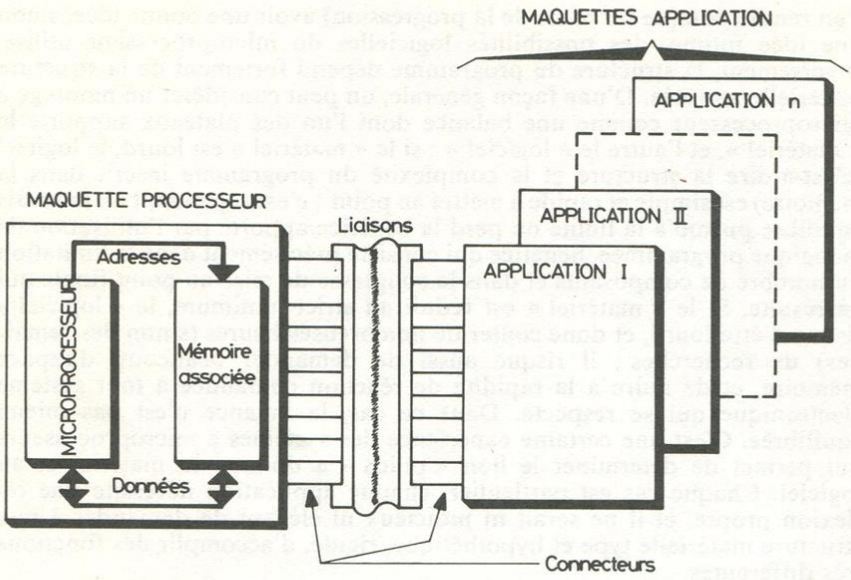


Fig. I. 1. Utilisation de la maquette processeur.

l'ensemble constituant un montage autonome à microprocesseur, obéissant à une fonction déterminée (horloge, synthétiseur, thermomètre, commande de machine-outil, etc.). On remarque que la maquette processeur est commune à chaque application, mais naturellement le programme (c'est-à-dire l'EPROM) change avec l'application.

II. 3. La maquette « mise au point »

La vie n'est pas aussi simple. En effet, l'ensemble « maquette application — maquette processeur » prend vie seulement à partir du moment où le contenu de l'EPROM (cerveau du système) n'est pas incohérent. Cette EPROM doit contenir un programme, et il se pose le problème de sa mise au point. En effet, un programme, nécessairement pour nous écrit en langage assembleur (ou machine), doit d'abord être pensé, puis rédigé sur une feuille de papier. Il est tout à fait improbable (car il y a peu de génies) qu'à ce stade il puisse être mis en EPROM, puis immédiatement fonctionner. Il est indispensable de disposer d'un système qui puisse permettre de tester tout ou partie d'un programme, et de corriger les erreurs inévitables qui se glissent lors de son élaboration.

Nous appelons ce système « maquette mise au point ». Cette maquette, dont nous parlons en détail plus loin, est destinée à être insérée, comme le

montre la figure I. 2., entre la maquette processeur et une maquette application.

Les fonctions sont les suivantes :

a) Elle contient 2 K octets de RAM destinés à simuler l'EPROM de 2 K octets de la maquette processeur. Les données sont entrées par l'intermédiaire d'un clavier situé sur une maquette dite *« maquette clavier-afficheur »*. L'ensemble de la figure 2 se comporte exactement comme l'ensemble de la figure 1, sauf que le programme se trouve en RAM, et est donc susceptible de toutes les modifications et mises au point utiles.

b) Les 2 K octets de RAM de la maquette mise au point peuvent être sauvegardés, par l'intervention d'un accumulateur, lors de la coupure accidentelle ou non de l'alimentation, grâce à l'utilisation de la technologie CMOS.

c) Ces mêmes 2 K octets de RAM peuvent aussi être sauvegardés sur la bande magnétique d'un magnétophone bon marché, grâce à l'interface cassette résident.

d) Enfin, les 2 K octets de RAM peuvent être transférés dans une EPROM 2716 (Intel) ou équivalent grâce à l'interface *programmateur d'EPROM* résident.

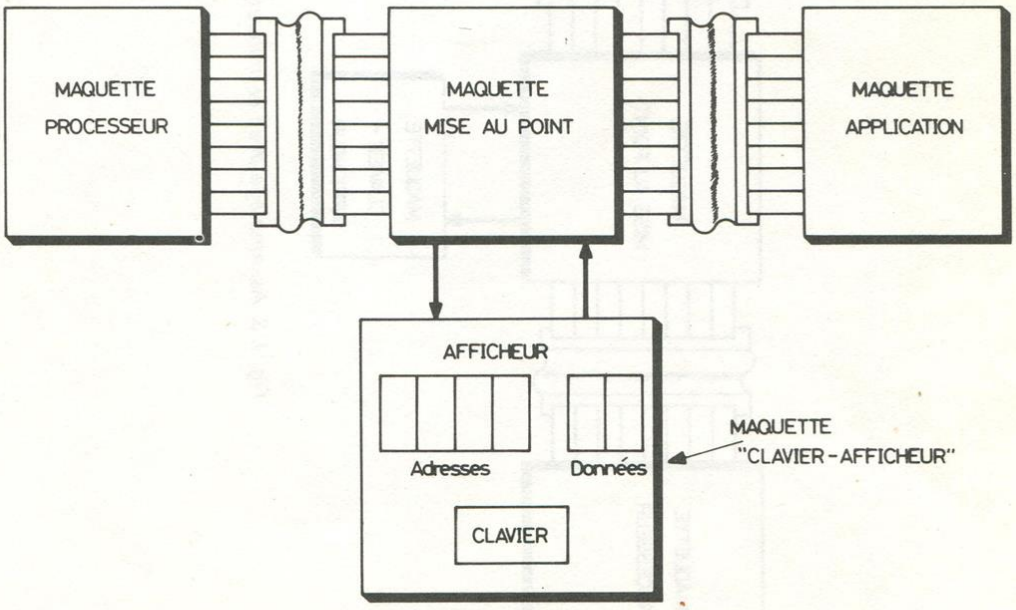


Fig. I. 2. Assemblage des maquettes

II. 4. La maquette « entrée/sortie »

Si vous remplacez la maquette application de la figure I . 2. par une maquette dite « entrée/sortie » permettant de communiquer avec le monde extérieur, vous obtenez le montage à microprocesseur de la figure I. 3. Ce

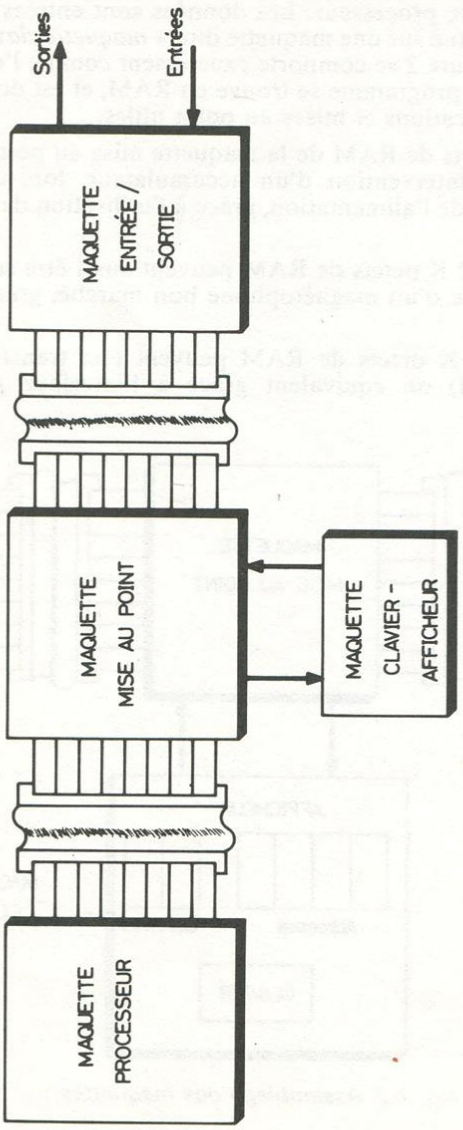


Fig. I. 3. Assemblage avec la maquette « entrée-sortie »

montage, qui n'a plus comme fonction essentielle de mettre au point une application finie, est tout simplement ce qu'on appelle couramment un micro-ordinateur. Ses entrées/sorties sont destinées à être couplées aux périphériques de votre choix, ou à commander et contrôler les éléments actifs de tout système à automatiser. Il est à noter que la maquette « clavier-afficheur » constitue elle-même un périphérique, et peut donc être traitée comme telle.

Il apparaît maintenant qu'il est illusoire de considérer les disciplines « électronique » et « informatique » comme constituant deux mondes à part : un électronicien désirant réaliser un système à microprocesseur est conduit à une incursion dans le domaine de l'informatique; inversement, un informaticien ayant le même but ou désirant utiliser son matériel de façon raisonnée et efficace, sera nécessairement conduit à s'introduire dans le monde de l'électronique.

III. Le microprocesseur utilisé

Il y a à peu près une dizaine de microprocesseurs 8 bits actuellement sur le marché. Il a bien fallu que nous en choisissions un, puisque nous proposons un système cohérent à réaliser pratiquement. On pourra de toutes façons trouver à redire sur notre choix. Mais il se trouve que, pour notre part, ce choix a été raisonné. Il faut dire que nous sommes professeurs d'électronique, et que nous sommes tout à fait insensibles aux pressions externes à nos convictions profondes. En effet, nous ne tenons aucun compte des rumeurs selon lesquelles tel ou tel « micro » serait plus « puissant » qu'un autre, ou plus « rapide », ou plus « performant », ou plus ce que l'on voudra. Nous avons la faiblesse de ne croire qu'en ce que nous avons expérimenté ; en outre, nous affirmons qu'il faut être très prudent lors de l'attribution d'un prix de « meilleure qualité » à un microprocesseur.

Donnons quelques lignes de justification sur le choix que nous avons fait du microprocesseur CDP 1802 (RCA) ; à noter que nous ne cherchons pas à prouver au lecteur le *bien fondé* de notre décision, mais seulement à exprimer notre façon de penser.

III. 1. L'architecture interne du CDP 1802

Donnée figure I. 4, elle présente un ensemble essentiel : l'aire des 16 registres R. Ces registres R sont numérotés de 0 à F (notation hexadécimale) ; par exemple, le registre de numéro 9 est noté R(9). Chacun de ces registres a une capacité de 16 bits, et doit donc pouvoir être divisé en deux parties égales, lors par exemple d'opérations de transfert vers l'accumulateur D qui est de capacité 8 bits : les 8 bits de poids fort du registre R(W) — W représentant un nombre de 0 à F — sont notés : R(W).1, et les 8 bits de poids faible : R(W).0. Quels sont les rôles de ces registres ?

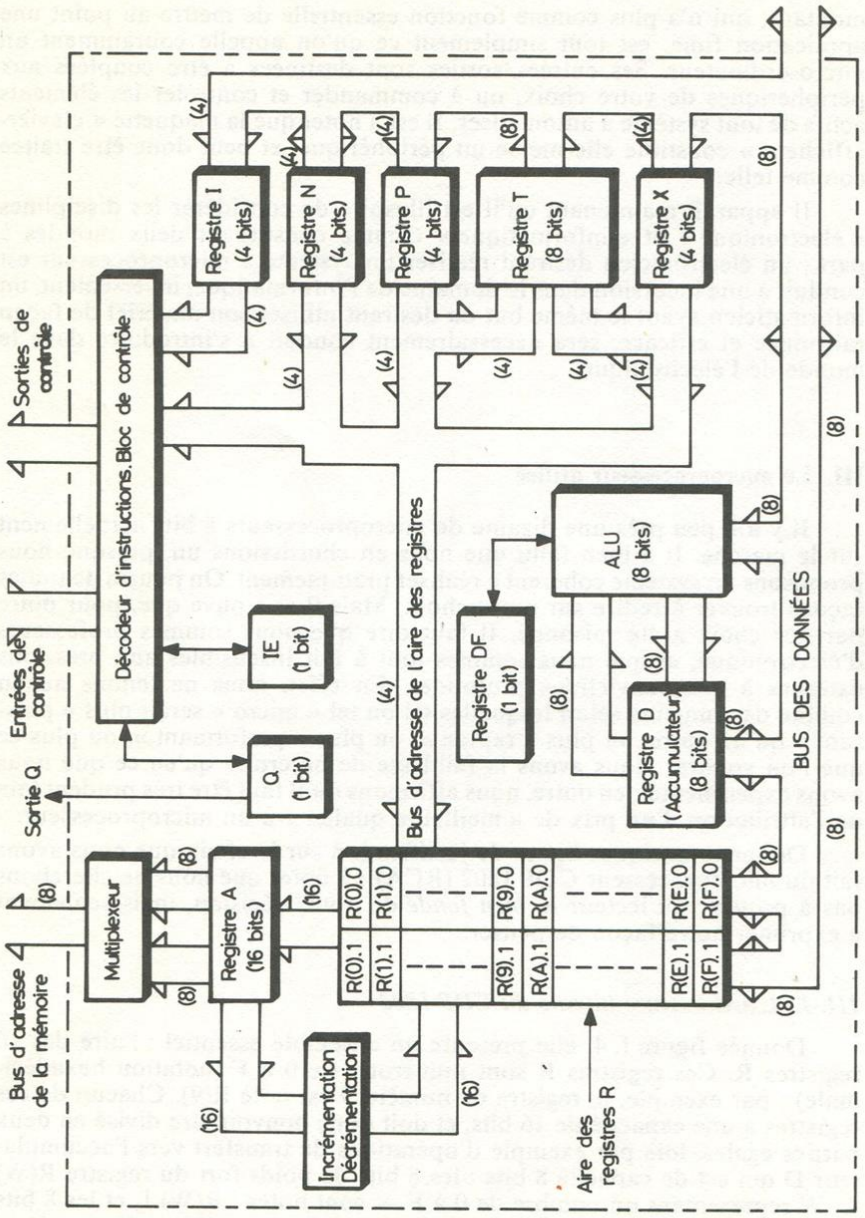


Fig. I. 4. Architecture interne du CDP 1802

a) Rôle de *compteur de boucles* ou de *case mémoire.* Le registre R ayant ce rôle attribué, est désigné par le registre de 4 bits N (4 bits étant nécessaires et suffisants pour désigner un registre R parmi 16). Ce registre R est alors noté R(N). Le contenu du registre de 4 bits N est défini par le code même de toute instruction ayant trait au registre R(N). Par exemple — voyez le jeu d'instructions — l'instruction DEC a pour effet de décrémenter le registre R(N) ; son code est 2 N, N étant le caractère hexadécimal désignant le registre R(N) : l'instruction de code 2E a pour effet de décrémenter le registre R(E). Voyons un autre exemple : l'instruction GLO a pour effet de transférer les 8 bits de poids faible de R(N) vers l'accumulateur D, ce qui est noté R(N).0 → D. Le code de cette instruction est 8 N ; entendons par là que l'instruction de code 81 par exemple a pour résultat suivant : R(1).0 → D.

b) Chaque registre R(W) peut avoir le rôle de *compteur de programme P.C.* Ceci est exceptionnel dans l'espèce des microprocesseurs ; en effet, habituellement, le P.C. est défini une bonne fois pour toutes par le constructeur, et est donc unique. Ce qui est unique ici, c'est que le programmeur peut à tout moment changer de P.C. (au cours d'un programme, par exemple pour faire un appel de sous-programme, ou pour toute autre raison). Le registre R à qui le rôle de P.C. est attribué, est désigné par le registre de 4 bits P. Ce registre R, donc le P.C., est alors noté R(P). Il est à noter que, après une remise à zéro par l'activation de l'entrée $\overline{\text{CLEAR}}$ du microprocesseur, P est remis automatiquement à zéro ainsi que le registre R(0) ; autrement dit, après une initialisation, c'est R(0) qui est P.C. et son contenu est zéro (tout programme commence à l'adresse zéro).

c) Chaque registre R(X) peut avoir le rôle de *pointeur de pile* (Stack Pointer en anglais) ou de *pointeur de toute autre zone mémoire.* Ceci est également inhabituel ; en effet, pour les autres microprocesseurs, le pointeur de pile est défini une bonne fois pour toutes. Dans notre cas, le pointeur de pile est désigné par le registre de 4 bits X. Le registre R ainsi désigné est noté R(X). En fait, nous considérons que cette architecture présente une symétrie séduisante et une certaine qualité esthétique. A la limite, ce pourrait être un critère de choix suffisant, pour nous, professeurs d'électronique qui cherchons à expliquer le microprocesseur aux autres.

III. 2. Les entrées et sortie directes du CDP 1802 : (voir brochage fig. I. 5)

De façon plus pragmatique, il est très intéressant de constater que ce microprocesseur présente une sortie directe Q dont l'état logique est imposé par programme (sauf lors de l'activation de $\overline{\text{CLEAR}}$ qui entraîne la remise à zéro de la sortie Q). Par ailleurs, 4 entrées « drapeau » (Flag en anglais) notées $\overline{\text{EF}}_1$, $\overline{\text{EF}}_2$, $\overline{\text{EF}}_3$, $\overline{\text{EF}}_4$ sont disponibles directement sur le boîtier du microprocesseur ; l'état logique de ces entrées peut être testé par programme par le biais des instructions de « branchement si ». L'existence de ces entrées et sortie directes peut considérablement simplifier la structure matérielle d'un système. De ce fait, l'interface cassette de la « maquette mise

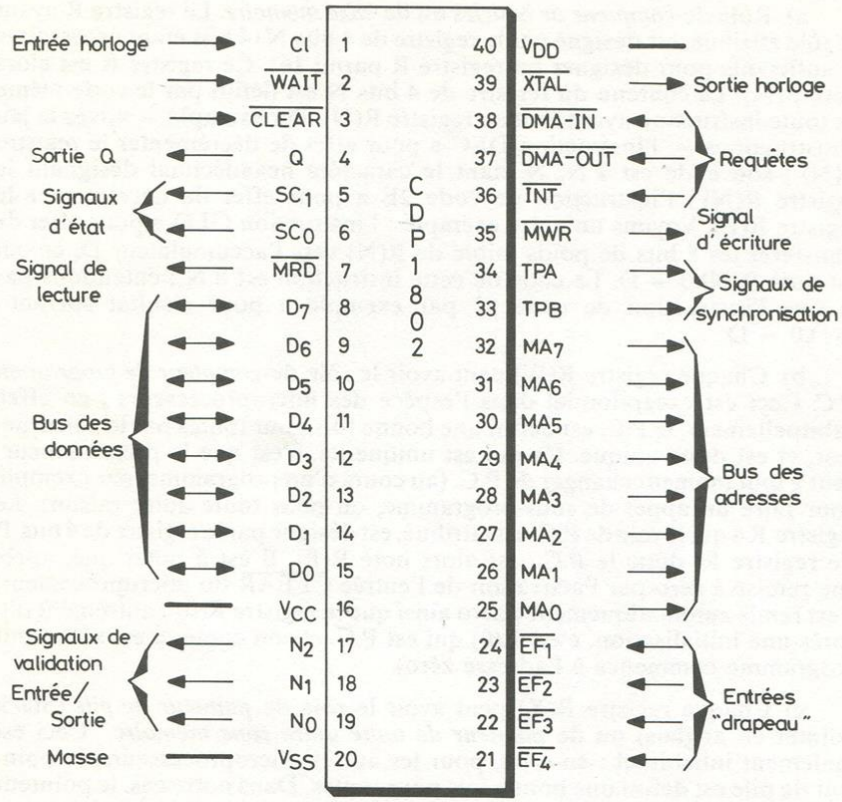


Fig. I. 5. Brochage du CDP 1802

au point » nécessite très peu de composants, car elle utilise la sortie Q. Il en est de même pour l'interfaçage avec le clavier de la « maquette clavier-afficheur » ; en effet, le clavier de 20 touches a une structure matricielle en 4 lignes et 5 colonnes, les 4 lignes étant directement reliées au 4 entrées drapeau EF_1^V.

III. 3. Les sorties N_2, N_1, N_0 du CDP 1802

Outre la sortie Q, le microprocesseur CDP 1802 présente 3 sorties N_2, N_1, N_0 activées seulement pendant l'exécution des instructions OUTPUT et INPUT. Le jeu d'instructions montre qu'il existe 7 instructions de sortie (et 7 instructions d'entrée) ; par exemple, lors de l'exécution de l'instruction OUT 5, les 3 sorties (N_2, N_1, N_0) sont au niveau 5 (décimal) c'est-à-dire (1,

0, 1) en binaire, autrement dit les lignes N_2, N_0 sont activées. Cela sert généralement à valider un boîtier de sortie (ou d'entrée) mais peut également être utilisé à déclencher une bascule, un compteur, un décodeur..., ce qui est très précieux comme nous le verrons lorsque nous parlerons du décodage du clavier.

III. 4. Les interruptions de programme du CDP 1802

Le microprocesseur utilisé présente trois entrées notées $\overline{\text{INT}}$, $\overline{\text{DMA-IN}}$ et $\overline{\text{DMA-OUT}}$. Ces trois entrées ont comme effet, lorsqu'elles sont activées, d'interrompre temporairement un programme en train de tourner. Dans quel but ?

a) *L'interruption par le canal $\overline{INT}$*

Voyons d'abord ce qui se passe lorsqu'un opérateur ou un système électronique active l'entrée $\overline{\text{INT}}$ (interrupt) d'un microprocesseur d'un autre type que celui utilisé par nous. Si l'entrée $\overline{\text{INT}}$ est activée, et si un certain bit d'autorisation d'interruption est mis (indiquant par là que le microprocesseur accepte de traiter l'interruption), le microprocesseur termine l'exécution de l'instruction en cours et s'engage dans un sous-programme (S.P.) d'interruption. Cependant, comme la requête d'interruption peut être établie à tout moment, c'est-à-dire à une époque où *l'état* du microprocesseur, soit le contenu de ses registres internes, est inconnu, et qu'il est nécessaire d'être en mesure de retourner au programme principal, le microprocesseur ne s'engage effectivement dans le sous-programme d'interruption qu'après avoir procédé à une sauvegarde du contexte dans lequel il était placé avant la requête de l'interruption. Cette sauvegarde consiste à stocker dans une RAM pointée par le pointeur de pile, le contenu de tous les registres internes (en particulier le compteur de programme, l'accumulateur ...). Cette sauvegarde prend du temps, et l'interruption n'est servie qu'au bout d'un certain délai pas forcément négligeable.

Dans le cas du CDP 1802, la procédure de sauvegarde de l'état de ses registres internes n'est pas automatique, donc doit être programmée par l'utilisateur. On pourrait croire qu'il s'agit d'un défaut, puisque cela demande un surcroît de travail et d'attention de la part du concepteur à qui il est demandé de décider lui-même de déterminer les registres dont il est nécessaire de sauvegarder le contenu. En réalité, cette liberté de décision permet, dans le cas précisément d'une application ne nécessitant pas la sauvegarde d'un grand nombre de registres, donc ne nécessitant pas beaucoup de temps, d'obtenir une réponse extrêmement rapide à une requête d'interruption, et par ce fait de ne pas annihiler le caractère d'urgence qu'elle présente. La rapidité de réaction à une interruption est encore augmentée du fait qu'il n'est pas nécessaire de sauvegarder le contenu du compteur de programme ; en effet, on se rappelle que nous disposons de *plusieurs* P.C. : par construction, c'est *R(1) qui est P.C. lors de l'exécution de la routine d'interruption.* En outre, *le rôle de pointeur de pile est automatique-*

ment affecté au registre R(2), et ceci toujours dans le but d'augmenter au maximum la rapidité de réaction du microprocesseur.

b) *l'interruption par le canal* $\overline{DMA}$

Les entrées $\overline{\text{DMA-IN}}$ et $\overline{\text{DMA-OUT}}$ (DMA comme Direct Memory Access, soit accès direct à la mémoire) servent, lorsqu'elles sont activées, à obtenir un accès direct à la mémoire en vue d'y écrire ($\overline{\text{DMA-IN}}$) une donnée ou de la lire (DMA-OUT). *Dans ces cas, le pointeur de la mémoire est R(0) ;* ce registre est un compteur qui s'incrémente à chaque fois qu'un cycle DMA a été exécuté.

Les lecteurs de notre précédent ouvrage déjà cité, auront apprécié l'existence de la broche $\overline{\text{DMA-IN}}$, puisqu'elle leur aura permis d'entrer un programme dans la mémoire associée au microprocesseur avec le concours d'un nombre extrêmement réduit de composants.

La broche $\overline{\text{DMA-OUT}}$ peut être utilisée à la lecture d'une zone mémoire destinée au rafraîchissement d'un écran de visualisation, ou à la synchronisation du microprocesseur sur une source d'impulsions issue du monde extérieur comme nous le verrons plus loin.

Ce rapide survol du CDP 1802 nous a permis peut-être de nous justifier quant à notre choix. Naturellement nous avons passé d'autres raisons sous silence. Comme vous le constatez, la description que nous venons de faire est incomplète, mais c'est au fur et à mesure de la progression de notre ouvrage que nous ferons plus ample connaissance avec ce microprocesseur.

Terminons ce chapitre par le tableau I.1 qui présente le jeu d'instructions du microprocesseur CDP 1802.

III. 5. Le jeu d'instructions du CDP 1802

Pour une lecture (et une utilisation) aisée du jeu d'instructions, il est nécessaire d'avoir quelques idées-clé en tête :

a) *Les notations utilisées*

Les opérations qui font référence à un octet situé dans la mémoire associée M, utilisent un registre R comme pointeur de cette mémoire. Ce registre R, qui contient l'adresse de l'octet à traiter, est soit le registre R(N), soit le registre R(X), soit le registre R(P), et ceci selon le type de l'instruction. L'octet situé dans la mémoire M est noté M(R(N)), M(R(X)), ou M(R(P)) selon qu'il est pointé par R(N), R(X), ou R(P). Par exemple, l'instruction STORE VIA N a pour effet suivant : D → M(R(N)) ; traduisez : l'octet situé dans l'accumulateur D est transféré (stocké) dans la mémoire M à l'adresse définie par le contenu de R(N). Autre exemple, l'instruction INPUT 1 exécute l'opération suivante : BUS → M(R(X)) ; BUS → D ; N lignes = 1. Traduisez en français : la sortie N_0 du microprocesseur est activée, ce qui valide un coupleur destiné à transmettre un octet du monde extérieur vers le système ; cet octet apparaît alors sur le BUS des données,

puis est transféré à la fois dans la mémoire M à l'adresse définie par le contenu de R(X), et dans l'accumulateur D.

b) *L'adressage immédiat*

Prenons encore un exemple. L'instruction LOAD IMMEDIATE (mnémonique : LDI) réalise l'opération suivante: M(R(P)) → D. Le code de LDI est F 8, et nous supposons (fig. I. 6) qu'il est inscrit à l'adresse n de

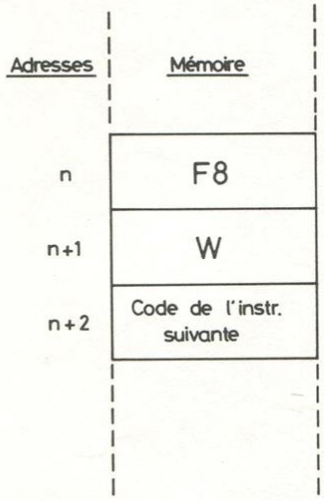


Fig. I. 6. Adressage immédiat

la mémoire M. Il y a un point essentiel à avoir en tête : le compteur de programme R(P) pointe, pendant le cycle d'exécution de toute instruction, la ligne mémoire *suivant* celle où se trouve inscrit le code de cette instruction (R(P) est en effet incrémenté à la fin du cycle de recherche). Par conséquent, au cours de *l'exécution* de l'instruction LDI, R(P) pointe la mémoire à l'adresse n+1, ce qui veut dire que M(R(P))=W ; ainsi le résultat de LDI est de transférer W dans l'accumulateur D. L'instruction LDI se termine par R(P)+1 → R(P), simplement pour que le P.C. pointe le code de l'instruction suivante situé à l'adresse n+2 de la mémoire, code qui sera transféré dans le microprocesseur lors du prochain cycle de recherche.

L'adresse du mot W à transférer dans D est immédiatement celle qui suit l'adresse où se trouve le code de l'instruction : c'est pour cela que le mode d'adressage de ce mot W est dit « immédiat ». A chaque fois que c'est le compteur de programme R(P) qui est utilisé pour l'adressage, nous avons affaire à un adressage immédiat.

Ceci a une conséquence qu'on ne peut se permettre d'oublier : si vous désirez demander au microprocesseur l'exécution d'une instruction nécessitant un adressage immédiat, il vous faudra lui indiquer non seulement le

code de cette instruction, mais immédiatement ensuite, l'*opérande* W à traiter. Il s'agit donc d'un type d'instruction à *deux* octets.

c) *Branchement long et branchement court*

Encore un exemple. L'instruction SHORT BRANCH (mnémonique BR) exécute l'opération suivante : M(R(P)) → R(P).0. Comme l'indique la figure I. 7, le code 30 de cette instruction est inscrit à l'adresse n de la

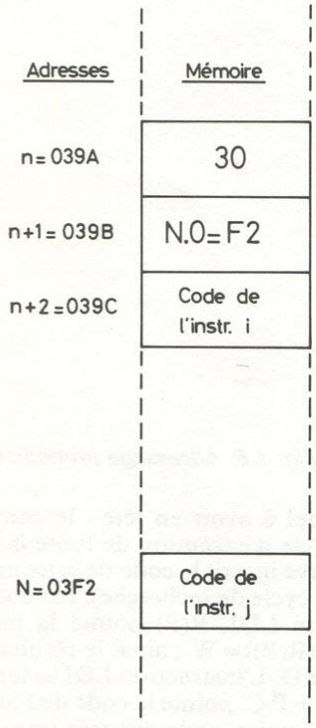


Fig. I. 7. Branchement court

mémoire M. L'adresse n, que nous rappelons être constituée de 16 bits c'est-à-dire 4 caractères hexadécimaux, est supposée égale à 039A. Lors de l'exécution de l'instruction BR, R(P) pointe la mémoire à l'adresse n + 1 = 039B ; par conséquent M(R(P)) = N.0 = F 2, et c'est donc N.0 = F 2 qui va être transféré dans les 8 bits de *poids faible* de R(P). Il s'agit donc d'un branchement à l'adresse 03F2, puisque le compteur de programme est affecté. On remarque que le poids fort (0 3) de R(P) n'est pas modifié, ce qui veut dire que l'amplitude du branchement (appelé alors branchement court) est limitée à une page mémoire (256 octets).

Regardons maintenant l'instruction LBR de branchement inconditionnel long. Son code C0 est inscrit à l'adresse n = 039A comme l'indique la figure I. 8. L'adresse d'arrivée N = 014B où l'on désire se brancher est

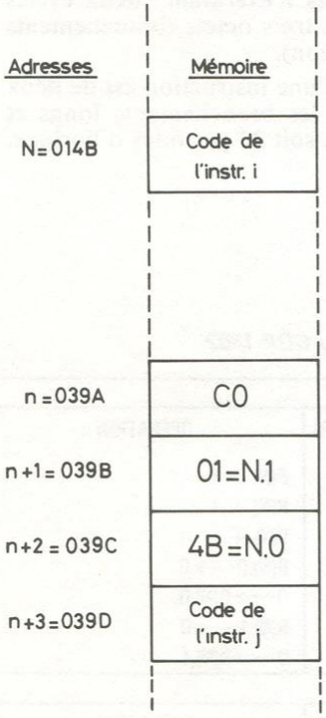


Fig. I. 8. Branchement long

« loin » derrière. L'opération réalisée lors de l'exécution de LBR est la suivante : $M(R(P)) \rightarrow R(P).1$; $M(R(P)+1) \rightarrow R(P).0$. Or, lors de l'exécution d'une instruction dont le code est inscrit à l'adresse n de la mémoire, le compteur de programme R(P) pointe la ligne d'adresse n + 1 ; par conséquent, la figure I. 8 montre que $M(R(P)) = N.1$ et $M(R(P)+1) = N.0$. L'effet de l'instruction LBR est ainsi de remplacer le contenu de R(P) par le mot de 16 bits N.1, N.0 : puisque le contenu du compteur de programme est affecté, il s'agit d'un branchement ; le branchement est dit « long » car le poids fort de R(P) est aussi affecté.

d) *Durée nécessaire à la réalisation d'une instruction*

Lorsqu'on raisonne sur les microprocesseurs, l'unité de temps à considérer est le *cycle machine*. Pour le CDP 1802, un cycle machine dure

8 périodes d'horloge (sauf le cycle spécial d'initialisation qui en dure 9). Soit dit en passant, à la fréquence maximale de travail de 6,4 MHz, un cycle dure 1,25 µs.

Pour la réalisation d'une instruction, il faut un cycle de *recherche* (du code de cette instruction), et *un ou deux* cycles *d'exécution* : deux cycles d'exécution sont requis pour les instructions à trois octets (branchements longs) et pour l'instruction NOP (pas d'opération).

Ainsi, la durée nécessaire à la réalisation d'une instruction est de deux cycles, soit 16 périodes d'horloge, sauf pour les branchements longs et l'instruction NOP où elle nécessite trois cycles, soit 24 périodes d'horloge.

Tableau I. 1. Jeu d'instruction du CDP 1802

Opérations sur les registres					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS (1)	CYCLES (2)	OPÉRATION
INCREMENT REG N	INC	1N	1	2	R(N) + 1
DECREMENT REG N	DEC	2N	1	2	R(N) − 1
INCREMENT REG X	IRX	60	1	2	R(X) + 1
GET LOW REG N	GLO	8N	1	2	R(N).0 ⟶ D
PUT LOW REG N	PLO	AN	1	2	D ⟶ R(N).0
GET HIGH REG N	GHI	9N	1	2	R(N).1 ⟶ D
PUT HIGH REG N	PHI	BN	1	2	D ⟶ R(N).1
Opérations de transfert entre l'accumulateur D et la mémoire.					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
LOAD VIA N	LDN	ON	1	2	M(R(N)) ⟶ D ; FOR N NOT 0
LOAD ADVANCE	LDA	4N	1	2	M(R(N)) ⟶ D ; R(N) + 1
LOAD VIA X	LDX	F0	1	2	M(R(X)) ⟶ D
LOAD VIA X AND ADVANCE	LDXA	72	1	2	M(R(X)) ⟶ D ; R(X) + 1
LOAD IMMEDIATE	LDI	F8	2	2	M(R(P)) ⟶ D ; R(P) + 1
STORE VIA N	STR	5N	1	2	D ⟶ M(R(N))
STORE VIA X AND DECREMENT	STXD	73	1	2	D ⟶ M(R(X)) ; R(X) − 1

Notes :
(1) : Cette colonne indique le nombre d'octets à programmer.
(2) : Cette colonne indique le nombre de cycles requis pour l'exécution (cycle de recherche compris) de l'instruction.

Tableau I. 1. Jeu d'instruction du CDP 1802 (suite)

Opérations logiques.

INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
OR	OR	F1	1	2	M(R(X)) OR D ⟶ D
OR IMMEDIATE	ORI	F9	2	2	M(R(P)) OR D ⟶ D ; R(P) + 1
EXCLUSIVE OR	XOR	F3	1	2	M(R(X)) XOR D ⟶ D
EXCLUSIVE OR IMMEDIATE	XRI	FB	2	2	M(R(P)) XOR D ⟶ D ; R(P) + 1
AND	AND	F2	1	2	M(R(X)) AND D ⟶ D
AND IMMEDIATE	ANI	FA	2	2	M(R(P)) AND D ⟶ D ; R(P) + 1
SHIFT RIGHT	SHR	F6	1	2	SHIFT D RIGHT, LSB (D) ⟶ DF, 0 ⟶ MSB (D)
SHIFT RIGHT WITH CARRY	SHRC	76	1	2	SHIFT D RIGHT, LSB (D) ⟶ DF, DF ⟶ MSB (D)
RING SHIFT RIGHT	RSHR				
SHIFT LEFT	SHL	FE	1	2	SHIFT D LEFT, MSB (D) ⟶ DF, 0 ⟶ LSB (D)
SHIFT LEFT WITH CARRY	SHLC	7E	1	2	SHIFT D LEFT, MSB (D) ⟶ DF, DF ⟶ LSB (D)
RING SHIFT LEFT	RSHL				

Opérations arithmétiques.

INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
ADD	ADD	F4	1	2	M(R(X))+D ⟶ DF, D
ADD IMMEDIATE	ADI	FC	2	2	M(R(P))+D ⟶ DF, D ; R(P) + 1
ADD WITH CARRY	ADC	74	1	2	M(R(X))+D+DF ⟶ DF, D
ADD WITH CARRY IMMEDIATE	ADCI	7C	2	2	M(R(P))+D+DF ⟶ DF, D ; R(P)+1
SUBSTRACT D	SD	F5	1	2	M(R(X))−D ⟶ DF, D
SUBSTRACT D IMMEDIATE	SDI	FD	2	2	M(R(P))−D ⟶ DF, D ; R(P)+1
SUBSTRACT D WITH BORROW	SDB	75	1	2	M(R(X))−D−(NOT DF) ⟶ DF, D
SUBSTRACT D WITH BORROW, IMMEDIATE	SDBI	7D	2	2	M(R(P))−D−(NOT DF) ⟶ DF, D ; R(P) + 1
SUBSTRACT MEMORY	SM	F7	1	2	D−M(R(X)) ⟶ DF, D
SUBSTRACT MEMORY IMMEDIATE	SMI	FF	2	2	D−M(R(P)) ⟶ DF, D ; R(P) + 1
SUBSTRACT MEMORY WITH BORROW	SMB	77	1	2	D−M(R(X))−(NOT DF) ⟶ DF, D
SUBSTRACT MEMORY WITH BORROW, IMMEDIATE	SMBI	7F	2	2	D−M(R(P))−(NOT DF) ⟶ DF, D ; R(P) + 1

Tableau I. 1. Jeu d'instruction du CDP 1802 (suite)

Instructions de branchement. (Branchement court).					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
SHORT BRANCH	BR	30	2	2	M(R(P)) ⟶ R(P).0
NO SHORT BRANCH (voir SKP)	NBR	38	2	2	R(P) + 1
SHORT BRANCH IF D=0	BZ	32	2	2	IF D=0, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF D NOT 0	BNZ	3A	2	2	IF D NOT 0, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF DF=1	BDF	33	2	2	IF DF=1, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF POS. OR ZERO	BPZ				
SHORT BRANCH IF EQUAL OR GREATER	BGE				
SHORT BRANCH IF DF=0	BNF	3B	2	2	IF DF=0, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF MINUS	BM				
SHORT BRANCH IF LESS	BL				
SHORT BRANCH IF Q=1	BQ	31	2	2	IF Q=1, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF Q=0	BNQ	39	2	2	IF Q=0, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF EFI=1	BI	34	2	2	IF EF1=1, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF EFI=0	BNI	3C	2	2	IF EF1=0, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF EF2=1	B2	35	2	2	IF EF2=1, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF EF2=0	BN2	3D	2	2	IF EF2=0, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF EF3=1	B3	36	2	2	IF EF3=1, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF EF3=0	BN3	3E	2	2	IF EF3=0, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF EF4=1	B4	37	2	2	IF EF4=1, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1
SHORT BRANCH IF EF4=0	BN4	3F	2	2	IF EF4=0, M(R(P)) ⟶ R(P).0 ELSE R(P) + 1

Tableau I. 1. Jeu d'instruction du CDP 1802 (suite)

Instructions de branchement. (Branchement long).

INSTRUCTION	NMÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
LONG BRANCH	LBR	C0	3	3	M(R(P)) ⟶ R(P).1 M(R(P)+1) ⟶ R(P).0
NO LONG BRANCH (voir LSKP)	NLBR	C8	3	3	R(P) + 2
LONG BRANCH IF D=0	LBZ	C2	3	3	IF D=0, M(R(P)) ⟶ R(P).1 M(R(P)+1) ⟶ R(P).0 ELSE R(P) + 2
LONG BRANCH IF D NOT 0	LBNZ	CA	3	3	IF D NOT 0, M(R(P)) ⟶ R(P).1 M(R(P)+1) ⟶ R(P).0 ELSE R(P) + 2
LONG BRANCH IF DF=1	LBDF	C3	3	3	IF DF=1, M(R(P)) ⟶ R(P).1 M(R(P)+1) ⟶ R(P).0 ELSE R(P) + 2
LONG BRANCH IF DF=0	LBNF	CB	3	3	IF DF=0, M(R(P)) ⟶ R(P).1 M(R(P)+1) ⟶ R(P).0 ELSE R(P) + 2
LONG BRANCH IF Q=1	LBQ	C1	3	3	IF Q=1, M(R(P)) ⟶ R(P).1 M(R(P)+1) ⟶ R(P).0 ELSE R(P) + 2
LONG BRANCH IF Q=0	LBNQ	C9	3	3	IF Q=0, M(R(P)) ⟶ R(P).1 M(R(P)+1) ⟶ R(P).0 ELSE R(P) + 2

Instructions de saut.

INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
SHORT SKIP (voir NBR)	SKP	38	1	2	R(P) + 1
LONG SKIP (voir NLBR)	LSKP	C8	1	3	R(P) + 2
LONG SKIP IF D=0	LSZ	CE	1	3	IF D=0, R(P) + 2 ELSE CONTINUE
LONG SKIP IF D NOT 0	LSNZ	C6	1	3	IF D NOT 0, R(P) + 2 ELSE CONTINUE
LONG SKIP IF DF=1	LSDF	CF	1	3	IF DF=1, R(P) + 2 ELSE CONTINUE
LONG SKIP IF DF=0	LSNF	C7	1	3	IF DF=0, R(P) + 2 ELSE CONTINUE
LONG SKIP IF Q=1	LSQ	CD	1	3	IF Q=1, R(P) + 2 ELSE CONTINUE
LONG SKIP IF Q=0	LSNQ	C5	1	3	IF Q=0, R(P) + 2 ELSE CONTINUE
LONG SKIP IF IE=1	LSIE	CC	1	3	IF IE=1, R(P) + 2 ELSE CONTINUE

Tableau I. 1. Jeu d'instruction du CDP 1802

Instructions de contrôle.

INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
IDLE	IDL	00	1	2	WAIT FOR DMA OR INTERRUPT ; M(R(0)) ⟶ BUS
NO OPERATION	NOP	C4	1	3	CONTINUE
SET P	SEP	DN	1	2	N ⟶ P
SET X	SEX	EN	1	2	N ⟶ X
SET Q	SEQ	7B	1	2	1 ⟶ Q
RESET Q	REQ	7A	1	2	0 ⟶ Q
SAVE	SAV	78	1	2	T ⟶ M(R(X))
PUSH X,P TO STACK	MARK	79	1	2	(X,P) ⟶ T ; (X,P) ⟶ M(R(2)) THEN P ⟶ X ; R(2) − 1
RETURN	RET	70	1	2	M(R(X)) ⟶ (X,P) ; R(X) + 1 1 ⟶ IE
DISABLE	DIS	71	1	2	M(R(X)) ⟶ (X,P) ; R(X) + 1 0 ⟶ IE

Instructions d'entrée/sortie

INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
OUTPUT 1	OUT 1	61	1	2	M(R(X)) ⟶ BUS ; R(X) + 1 N LINES = 1
OUTPUT 2	OUT 2	62	1	2	M(R(X)) ⟶ BUS ; R(X) + 1 N LINES = 2
OUTPUT 3	OUT 3	63	1	2	M(R(X)) ⟶ BUS ; R(X) + 1 N LINES = 3
OUTPUT 4	OUT 4	64	1	2	M(R(X)) ⟶ BUS ; R(X) + 1 N LINES = 4
OUTPUT 5	OUT 5	65	1	2	M(R(X)) ⟶ BUS ; R(X) + 1 N LINES = 5
OUTPUT 6	OUT 6	66	1	2	M(R(X)) ⟶ BUS ; R(X) + 1 N LINES = 6
OUTPUT 7	OUT 7	67	1	2	M(R(X)) ⟶ BUS ; R(X) + 1 N LINES = 7
INPUT 1	INP 1	69	1	2	BUS ⟶ M(R(X)) ; BUS ⟶ D N LINES = 1
INPUT 2	INP 2	6A	1	2	BUS ⟶ M(R(X)) ; BUS ⟶ D N LINES = 2
INPUT 3	INP 3	6B	1	2	BUS ⟶ M(R(X)) ; BUS ⟶ D N LINES = 3
INPUT 4	INP 4	6C	1	2	BUS ⟶ M(R(X)) ; BUS ⟶ D N LINES = 4
INPUT 5	INP 5	6D	1	2	BUS ⟶ M(R(X)) ; BUS ⟶ D N LINES = 5
INPUT 6	INP 6	6E	1	2	BUS ⟶ M(R(X)) ; BUS ⟶ D N LINES = 6
INPUT 7	INP 7	6F	1	2	BUS ⟶ M(R(X)) ; BUS ⟶ D N LINES = 7

Chapitre 2

CONCEPTION ET RÉALISATION PRATIQUE DE LA MAQUETTE « PROCESSEUR »

I. Organisation de l'espace mémoire

Il y a deux façons d'utiliser la maquette processeur :

1. Soit en la connectant directement à une maquette « application » (voir la figure I. 1). La maquette « processeur » contient alors nécessairement une EPROM dont le contenu est le programme de « gestion » de l'application envisagée. L'ensemble forme un système obéissant à une fonction bien définie. Nous dirons alors que nous sommes dans le MODE APPLICATION.

2. Soit en la connectant à une maquette « application » (ou à la maquette « entrée/sortie »), mais par l'intermédiaire de la maquette « mise au point » (voir les figures I. 2 et I. 3). Nous dirons alors que nous sommes dans le MODE MISE AU POINT.

La *maquette processeur* contient une EPROM de 2 K octets et une RAM de 128 octets. Il se pose la question de savoir dans quelles zones mémoire (comprises naturellement entre les adresses 0000 et FFFF) doivent se situer ces boîtiers. Pour répondre à cette question, il faut savoir qu'après activation de l'entrée $\overline{\text{CLEAR}}$ du microprocesseur, le compteur de programme (alors registre R(0)) contient l'adresse 0000. Par conséquent, dans le *mode application,* l'EPROM de 2 K octets doit nécessairement se situer dans l'espace mémoire compris entre les adresses 0000 et 07FF (fig. II. 1).

Dans le *mode mise au point,* on peut facilement comprendre que le programme, destiné à être contenu dans l'EPROM 2 K, n'est pas encore mis au point. Par conséquent il faut qu'après activation de l'entrée $\overline{\text{CLEAR}}$ du microprocesseur, cette EPROM ne *soit pas* sélectionnée, donc qu'elle réside *ailleurs* que dans l'espace compris entre 0000 à 07FF. Nous avons décidé (fig. II. 2) que dans le mode mise au point, cette EPROM 2 K serait située dans l'espace mémoire compris entre les adresses 1000 et 17FF.

Nous avons par ailleurs décidé que la RAM de 128 octets résiderait, qu'on soit dans le mode application ou le mode mise au point, dans la zone mémoire comprise entre les adresses 0C00 et 0C7F (voir figures II. 1 et II. 2).

— Remarque !
la Ram d'application
à la même position
sur les deux
maquettes.

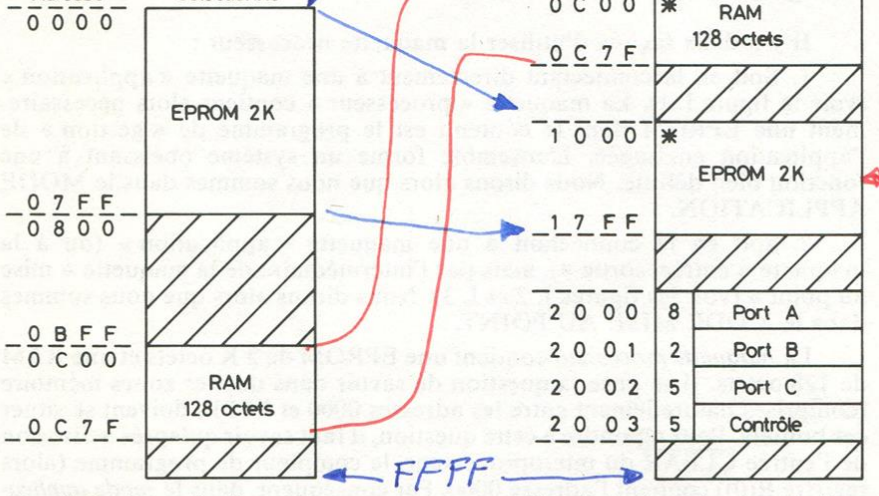


Fig. II. 1. Organisation de l'espace mémoire : maquette « processeur » en mode application

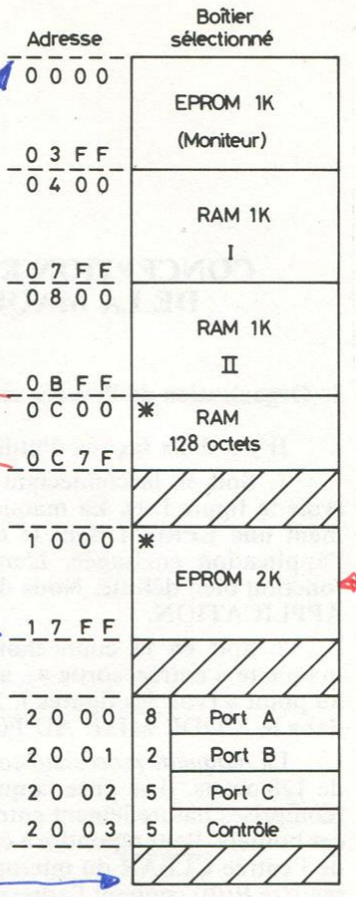


Fig. II. 2. Organisation de l'espace mémoire : maquette « processeur » en mode mise au point (boitiers mémoire de la maquette processeur)*

Les brochages de la plupart des circuits intégrés utilisés au cours de cet ouvrage sont donnés à la figure II. 3. Les boîtiers mémoire utilisés pour la maquette processeur sont :

1. Une mémoire 2716 de chez INTEL (ou équivalente) qui possède une entrée $\overline{CE}$ (Chip Enable) active au niveau bas. Si $\overline{CE}=1$, le boîtier est en haute impédance c'est-à-dire non sélectionné, et si $\overline{CE}=0$ le boîtier est

sélectionné. C'est en agissant sur $\overline{CE}$ qu'on peut placer cette mémoire dans tel ou tel espace.

2. Une mémoire RAM CDP 1823 de capacité 128 octets qui possède une pléthore de CS et $\overline{CS}$, notés CS_1, $\overline{CS_2}$, $\overline{CS_3}$, CS_4 et $\overline{CS_5}$. Le boîtier est sélectionné si et seulement si $CS_1 = 1$ *et* $\overline{CS_2} = 0$ *et* $\overline{CS_3} = 0$ *et* $CS_4 = 1$ *et* $\overline{CS_5} = 0$. Nous avons imposé $\overline{CS_2}$ et $\overline{CS_3}$ constamment à 0 pour limiter notre liberté d'action à la commande des entrées CS_1, CS_4 et $\overline{CS_5}$.

Soient maintenant A_{11}, A_{10}, ..., A_1, A_0 les 12 bits de poids faible des 16 bits d'adresse disponibles. Il faut se rendre compte que, pour placer l'EPROM 2716 dans la zone mémoire aux adresses 0000 à 07FF (mode application), il suffit de relier la ligne A_{11} à l'entrée $\overline{CE}$ de l'EPROM. En effet, on remarque alors que $\overline{CE} = 0$ (donc la mémoire validée) pour des adresses comprises entre 0000 à 07FF, et que $\overline{CE} = 1$ (donc la mémoire non sélectionnée) pour des adresses comprises entre 0800 et 0FFF. *On remarque aussi* une chose dont il faudra tenir compte selon l'application envisagée : le fait de relier simplement la ligne A_{11} à l'entrée $\overline{CE}$ de l'EPROM rend indifférent l'état logique des 4 bits A_{15}, A_{14}, A_{13}, et A_{12} quant à la sélection du boîtier EPROM. Cela veut dire que l'EPROM est également située dans l'espace mémoire aux adresses comprises entre 8000 et 87FF par exemple. Or si précisément, nous désirons développer une application pour laquelle l'espace 8000-87FF est déjà « pris », il deviendra nécessaire de prévoir une logique de décodage telle que $\overline{CE}$ soit à l'état bas si $A_{11} = 0$ *et* si $A_{15} = 0$.

Mais cette logique de décodage n'intervient éventuellement que sur une maquette application ; par conséquent, sur la maquette processeur, l'entrée $\overline{CE}$ de l'EPROM est laissée « libre » et disponible sur le connecteur.

Quant à la RAM de la maquette processeur, les entrées CS_1 et CS_4 sont reliées respectivement aux lignes d'adresse A_{10} et A_{11}. L'entrée $\overline{CS_5}$ est laissée « libre » et disponible sur le connecteur pour les mêmes raisons que celles énoncées précédemment. Lorsque la RAM est sélectionnée, c'est-à-dire si $\overline{CS_5}$ est au niveau bas, et si nous ne tenons pas compte de l'éventuelle influence des 4 bits A_{15}, A_{14}, A_{13} et A_{12}, on s'aperçoit qu'elle réside en particulier aux adresses comprises entre 0C00 et 0C7F.

II. Le schéma synoptique de la maquette processeur, dite VILÉMIO 1

Donné figure II. 4, il présente les centres d'intérêt suivants :

II. 1. La génération des bits d'adresse A_{11}, A_{10}, A_9 et A_8

Le bus d'adresse du microprocesseur est constitué de 8 fils. Cela ne veut pas dire que la transmission d'une adresse de 16 bits est impossible ; en effet, pour véhiculer 16 bits sur 8 fils, il suffit d'en véhiculer 8, *puis* les 8 autres. On dit alors que le bus d'adresse est multiplexé. Examinons la figure II. 5 qui représente les chronogrammes des signaux TPA, TPB et de

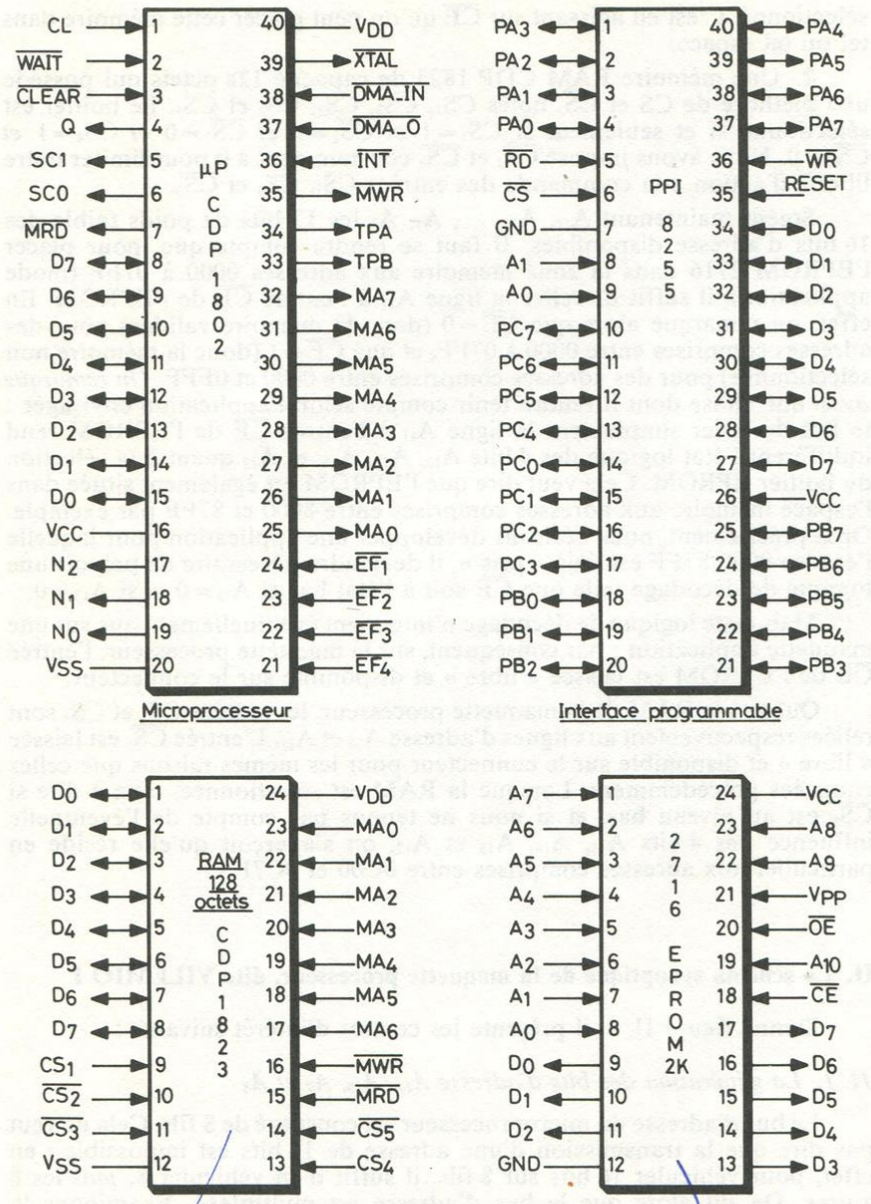
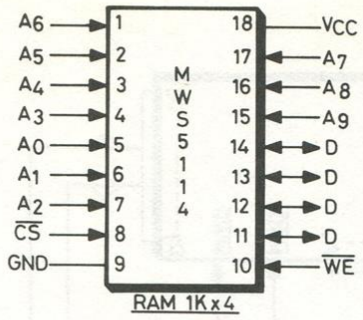
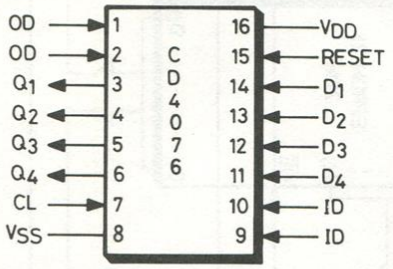


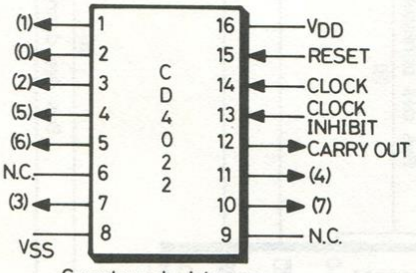
Fig. II. 3. Brochage des principaux



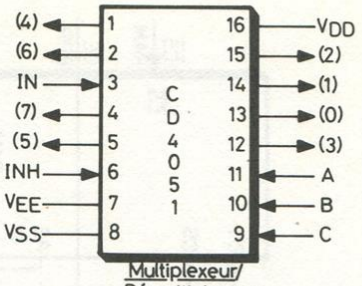
RAM 1K x 4



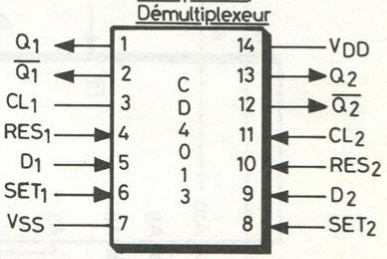
Registre 4 bits



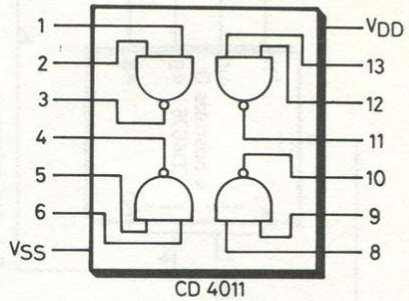
Compteur de Johnson



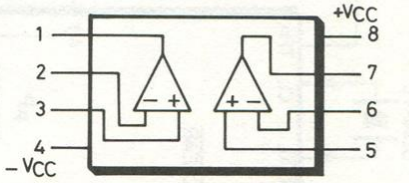
Multiplexeur/
Démultiplexeur



Registre 2 bits



CD 4011
Portes NAND



TL 072 CP
Double Amplif. Op.

circuits intégrés utilisés

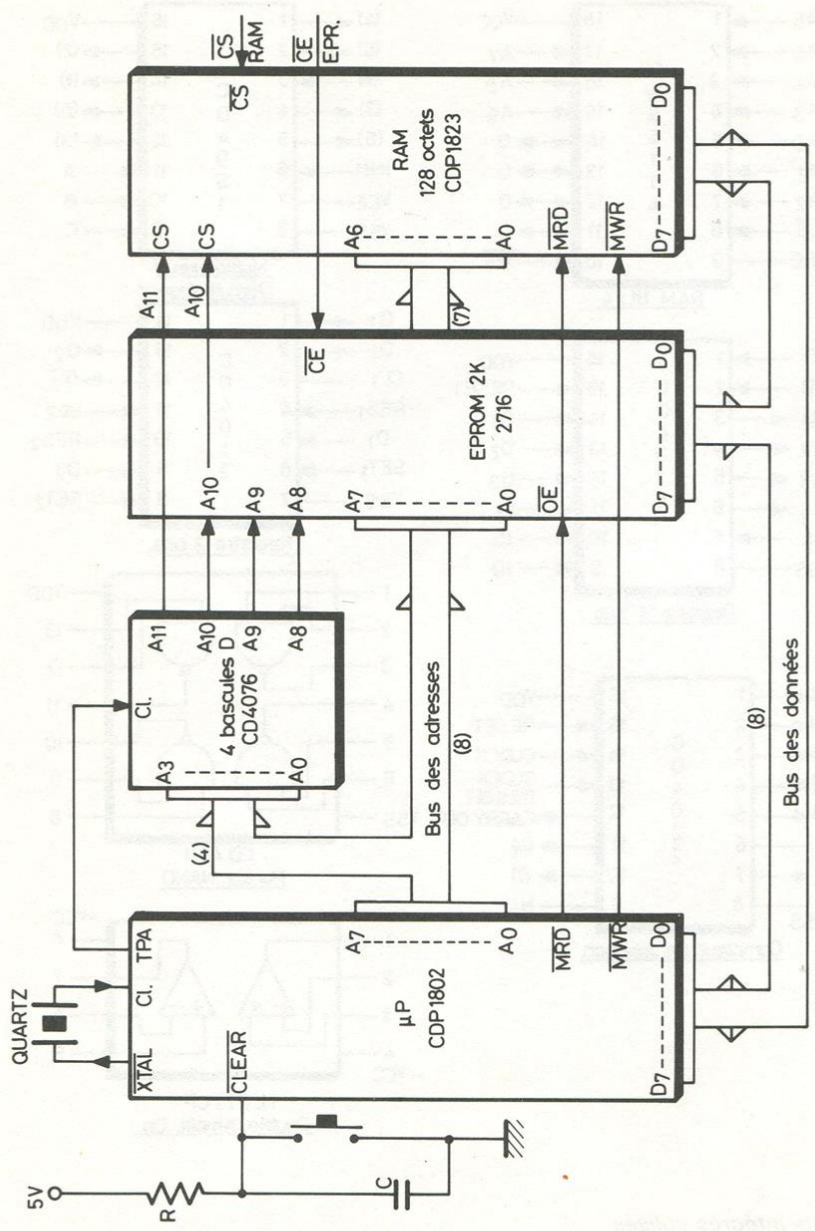


Fig. II. 4. Schéma synoptique de la maquette processeur

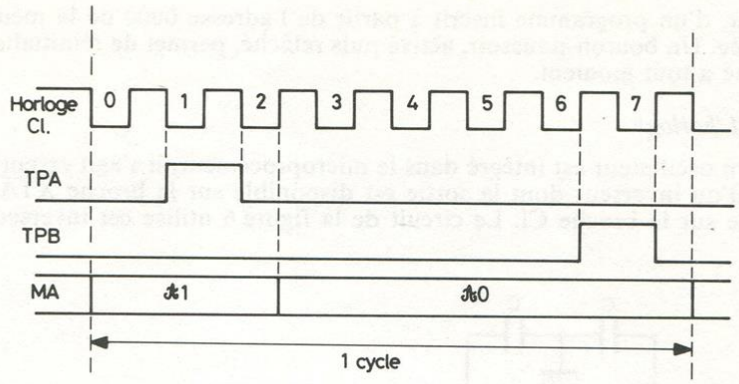


Fig. II. 5. Chronogrammes des signaux TPA, TPB et de l'état des lignes d'adresse MA

l'état des 8 lignes d'adresse MA relativement à l'horloge Cl. Les 8 périodes d'horloge constituent un *cycle machine* pendant lequel il apparaît que les 8 bits de *poids fort* notés $\mathcal{A}$ .1 d'une adresse à transmettre, sont présents sur les 8 lignes MA en début du cycle et en particulier lorsque *TPA = 1*. Lorsque les 8 bits de *poids faible* notés $\mathcal{A}$ .0 de l'adresse à transmettre sont présents sur les 8 lignes MA, *TPA = 0*.

Il est simple à comprendre que si $\mathcal{A}$ .0 est présent sur MA, $\mathcal{A}$ .1 n'y est plus : il faut donc mémoriser les bits utiles de $\mathcal{A}$ .1 dans un registre si l'on veut pouvoir adresser complètement la mémoire. C'est le rôle du registre de 4 bits CD 4076 constitué par 4 bascules D déclenchées simultanément par le signal TPA, de mémoriser les 4 bits d'adresse A_{11}, A_{10}, A_9 et A_8 nécessaires à, l'adressage de la mémoire EPROM et de la RAM de la maquette processeur.

Indiquons que le brochage du registre CD 4076 est donné figure II. 3. Les entrées ID (« Input Disable » comme « Entrée Interdite »), activées par un niveau 1, mettent l'entrée du registre en haute impédance. Les entrées OD (« Output Disable »), activées par un niveau 1, mettent la sortie de registre en haute impédance, nous avons mis les entrées ID et OD au niveau 0.

L'entrée RESET est aussi constamment placée au niveau 0 ; et c'est un front montant sur l'entrée Cl reliée à TPA qui déclenche le registre.

II. 2. L'entrée $\overline{CLEAR}$ du microprocesseur

Lors de la mise sous tension de la maquette processeur, l'entrée $\overline{CLEAR}$ est automatiquement activée (niveau 0) car la tension aux bornes du condensateur ne peut passer instantanément de 0 à 5 V. Au bout d'un délai de l'ordre de plusieurs fois la constante de temps R.C, l'entrée $\overline{CLEAR}$ atteint le niveau logique 1, ce qui fait démarrer l'exécution par le micropro-

cesseur, d'un programme inscrit à partir de l'adresse 0000 de la mémoire associée. Un bouton-poussoir, activé puis relâché, permet de réinitialiser le système à tout moment.

II. 3. L'horloge

Un oscillateur est intégré dans le microprocesseur, il s'agit essentiellement d'un inverseur dont la sortie est disponible sur la broche $\overline{\text{XTAL}}$, et l'entrée sur la broche Cl. Le circuit de la figure 6 utilise cet inverseur en

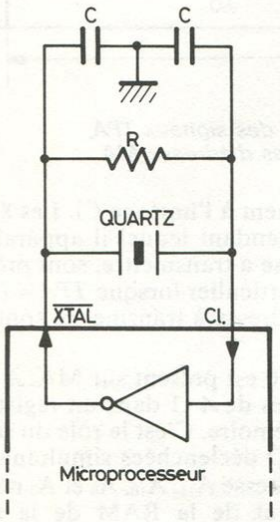


Fig. II. 6. Schéma de l'horloge

réalisant un oscillateur à quartz : la résistance R polarise l'inverseur dans sa zone linéaire, et les condensateurs C (bien que souvent inutiles) améliorent la stabilité en fréquence.

Quelle fréquence choisir ?

Le constructeur indique qu'on peut choisir dans la gamme de 0 à 3,2 MHz pour une tension d'alimentation de 5 V (la gamme s'étend à 6,4 MHz pour une tension d'alimentation de 10 V, ce qui n'est pas notre cas). Il est certain qu'on a toujours intérêt à travailler le plus vite possible (dans le domaine des microprocesseurs). Cependant, nous conseillons une fréquence de l'ordre de 2 MHz pour éviter les problèmes de résonances parasites dans les lignes de transmission des signaux ; par ailleurs, si dans une application de votre choix, vous désirez utiliser des boîtiers nécessitant la sortie $\overline{\text{XTAL}}$ pour fonctionner, vous constaterez que la fréquence maximale de travail pour ces boîtiers est souvent précisément

de 2 MHz. Toutefois ce choix n'a rien d'impératif (sauf pour l'utilisation du programme moniteur : voir plus loin).

III. Le schéma de câblage de la maquette processeur

Il est donné figure II. 7.

IV. Réalisation pratique de la maquette processeur

IV. 1. Considérations générales

Lorsqu'un électronicien aborde le domaine des microprocesseurs et désire passer à la réalisation pratique des systèmes qu'il conçoit lui-même, il doit réfléchir à la technique de fabrication qu'il compte utiliser. Les techniques fiables et bien connues sont les suivantes :

a) La technique du « wrapping ». Il s'agit dans un premier temps de placer les supports des circuits intégrés, et les picots devant supporter les éléments passifs (résistances, condensateurs, transistors...), sur une plaque isolante percée de multiples trous au pas de 0,1 pouce (2,54 mm) destinés à cet effet. Dans un deuxième temps, les diverses connexions sont réalisées avec du fil « volant » à l'aide d'un outil spécial appelé « outil à wrapper ». Cet outil, constitué d'une tige creuse contenant le fil destiné aux connexions, doit venir s'enficher sur les picots ou les broches des supports des circuits. La rotation imprimée à l'outil entraîne une rotation, en même temps que le serrage du fil autour des picots ou broches ; ceux-ci ayant une section carrée, le serrage du fil provoque son dénudage : la connexion est faite.

Cette technique est très fiable et donne d'ailleurs de meilleurs résultats qu'un circuit imprimé quant à la sensibilité aux chocs et vibrations ; en effet, rappelons que cette technique est utilisée lors de la fabrication des systèmes électroniques destinés aux satellites.

b) La technique du circuit imprimé

c) La technique qui combine les deux techniques précédentes

En fait, nous pensons que la technique du wrapping combinée à celle du circuit imprimé est la plus pratique pour un amateur isolé qui travaille seul.

Cependant, pour des raisons de reproductibilité (et donc d'édition de ce livre), nous avons été contraints à utiliser la technique du circuit imprimé, c'est-à-dire à concevoir son dessin.

Le circuit imprimé de la maquette processeur est *double* face. La figure II. 8 montre le circuit du côté composants, c'est-à-dire le dessin du circuit tel qu'on le voit, la maquette posée à l'endroit sur une table. La figure II. 9 montre le circuit du côté que l'on appellerait « côté cuivre » si

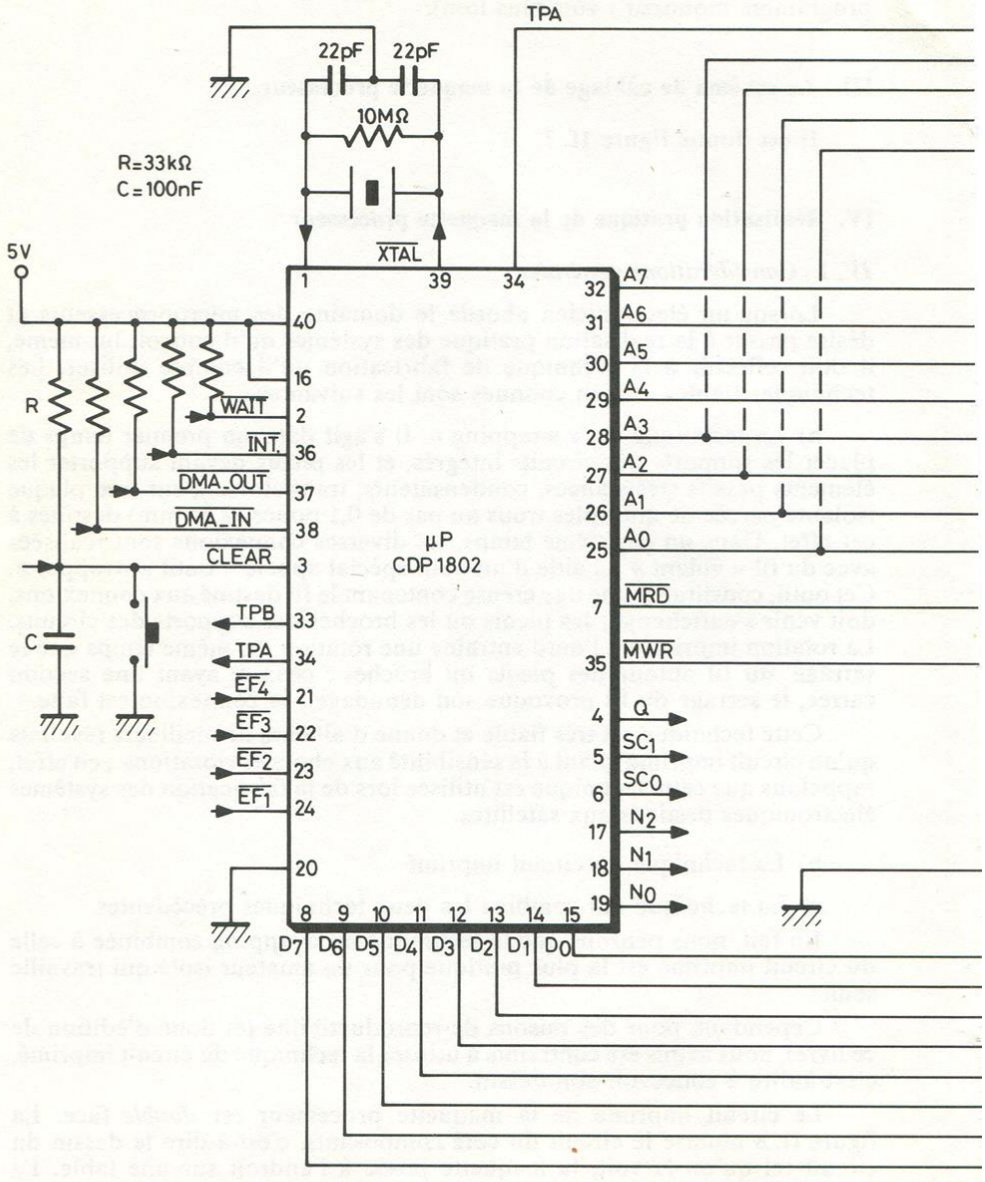
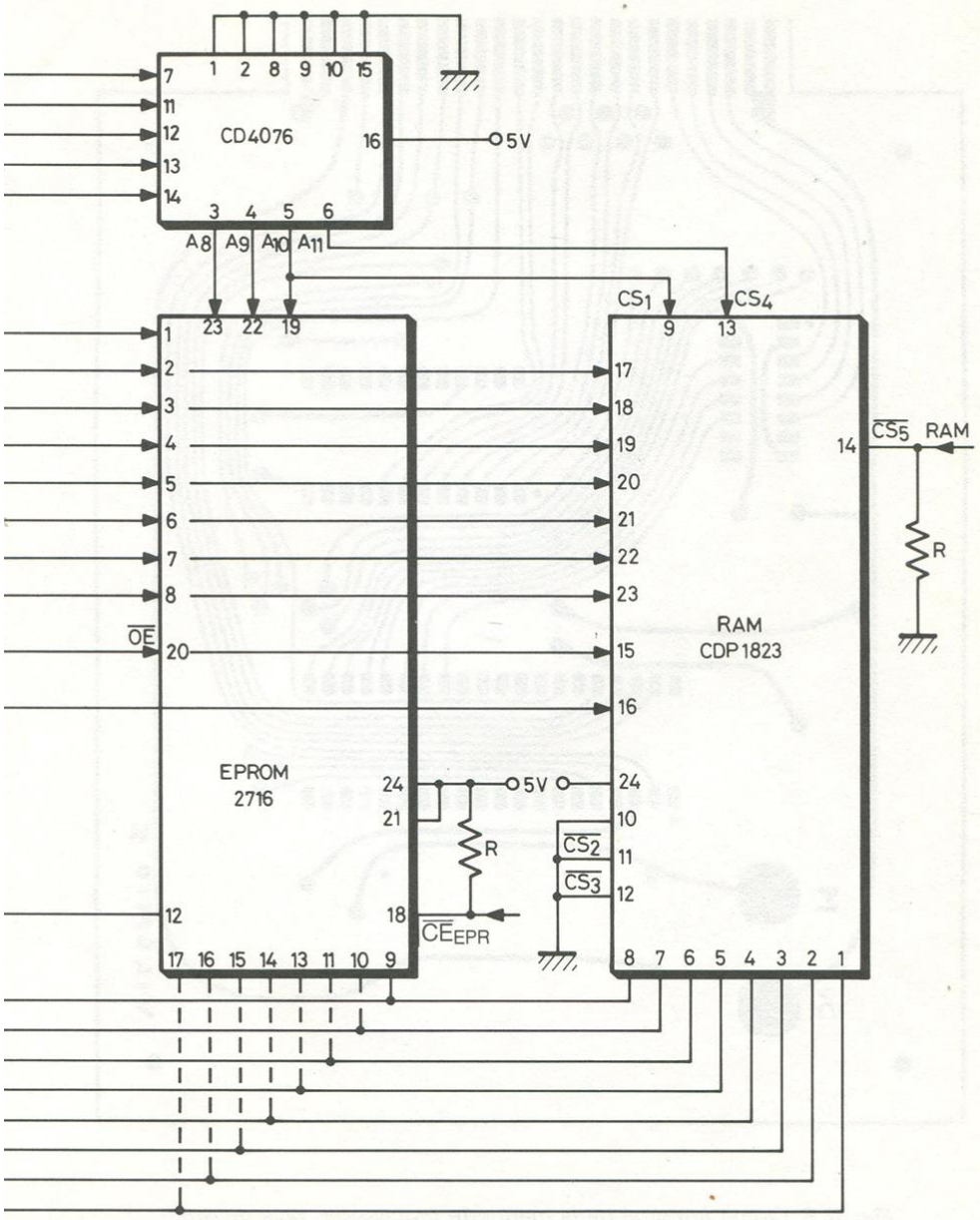


Fig. II. 7. Schéma de câblage



de la maquette processeur

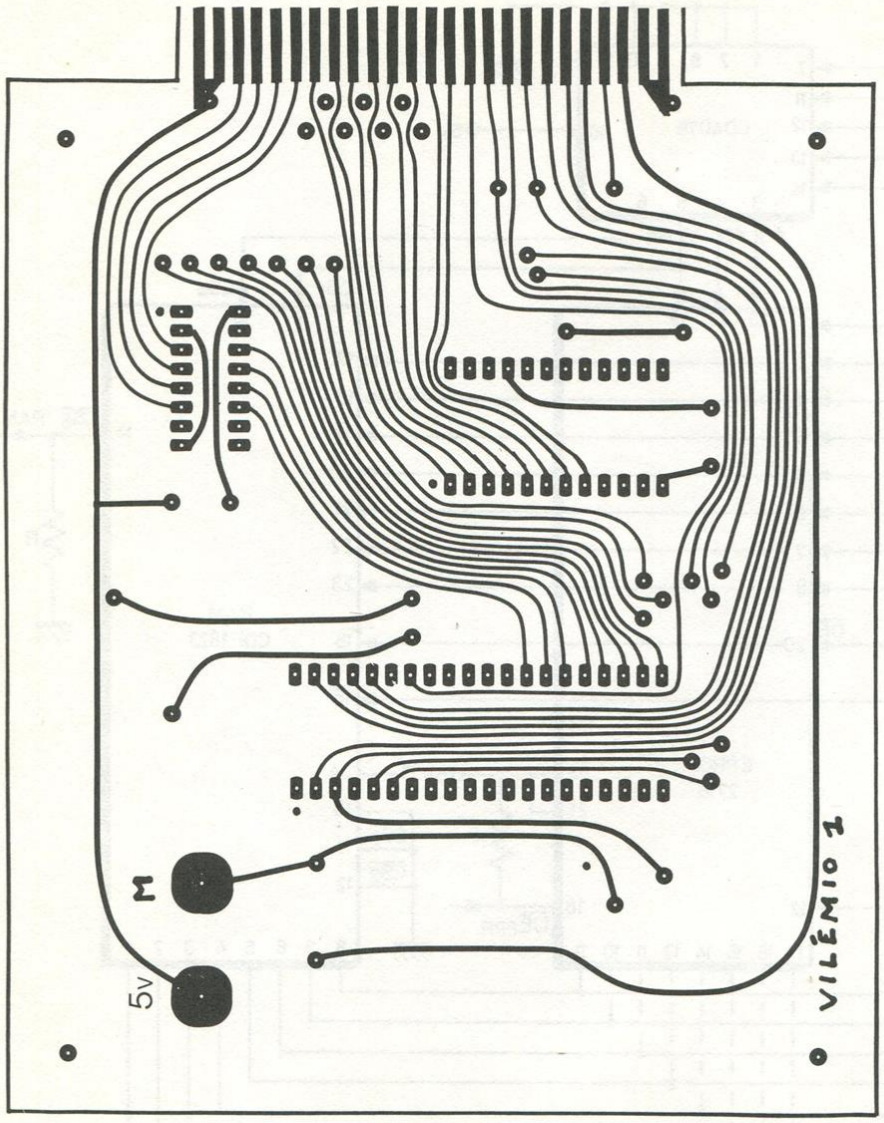


Fig. II. 8. Circuit imprimé de la maquette processeur, face dessus (côté composant)

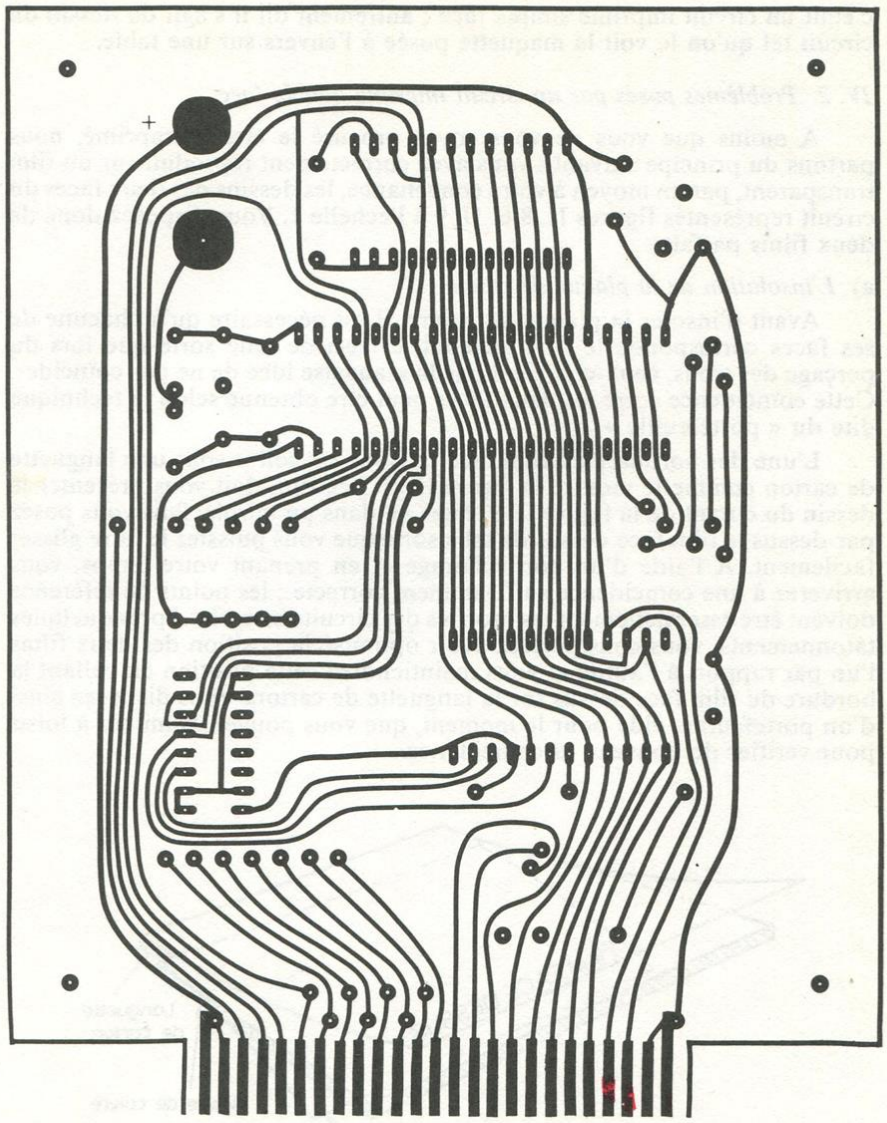


Fig. II. 9. Circuit imprimé face dessous (côté cuivre)

c'était un circuit imprimé simple face ; autrement dit il s'agit du dessin du circuit tel qu'on le voit la maquette posée à l'envers sur une table.

IV. 2. Problèmes posés par un circuit imprimé double face

A moins que vous ne vous soyez procuré le circuit imprimé, nous partons du principe suivant : vous avez correctement reproduit sur un film transparent, par un moyen à votre convenance, les dessins des deux faces du circuit représentés figures II. 8 et II. 9 à l'échelle 1. Vous disposez donc de deux films parfaits.

a) *L'insolation de la plaque de cuivre*

Avant d'insoler la plaque de cuivre, il est nécessaire qu'à chacune de ses faces corresponde le film adéquat, et ceci de telle sorte que lors du perçage des trous, ceux-ci n'aient pas la mauvaise idée de ne pas coïncider. Cette coïncidence entre les deux faces peut être obtenue selon la technique dite du « portefeuille ».

L'une des bordures du film face dessous est collée sous une languette de carton comme le montre la figure II. 10 ; ce film doit vous présenter le dessin du circuit de la figure II. 9, mais vu dans un miroir. Puis vous posez par-dessus le film face dessus de telle sorte que vous puissiez le faire glisser facilement. A l'aide d'un bon éclairage et en prenant votre temps, vous arriverez à une coïncidence suffisamment correcte ; les points de référence doivent être essentiellement les broches des circuits intégrés. Après quelques tâtonnements, vous considérerez avoir optimisé la position des deux films l'un par rapport à l'autre, et vous maintiendrez cette position en collant la bordure du film face dessus sur la languette de carton. Vous disposez ainsi d'un portefeuille, vide pour le moment, que vous pouvez examiner à loisir pour vérifier de nouveau la coïncidence.

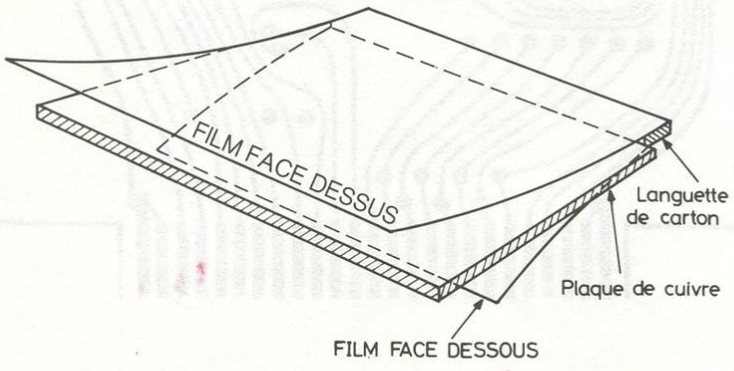


Fig. II. 10. Technique du « portefeuille »

Pour insoler la plaque de cuivre double face, il faut l'insérer dans le portefeuille que vous venez de confectionner, presser le tout et l'exposer aux U.V. Si vous avez peu de moyens techniques, par exemple si vous ne pouvez insoler qu'une face à la fois, il est nécessaire naturellement de retourner le portefeuille sans modifier la position de la plaque de cuivre par rapport aux films ; vous y parviendrez en fixant les films sur la plaque de cuivre avec du ruban adhésif.

b) *Les trous métallisés*

Lorsqu'on achète un système électronique « tout fait » monté sur une plaque de cuivre double face, ou que l'on fait réaliser un circuit imprimé double face par une société spécialisée, on constate l'existence de trous métallisés. Il s'agit d'un trou dont la paroi a été métallisée pour permettre une liaison électrique entre les deux faces. Ces trous sont soit vides, soit pleins. Dans ce dernier cas, ils sont destinés à recevoir les broches du support d'un circuit intégré par exemple. Il faut remarquer que la métallisation des trous permet de ne souder que d'un seul côté (le côté face dessous).

Nous n'avons pas trouvé le procédé permettant à un amateur de métalliser des trous. Nos trous métallisés seront donc constitués de la façon suivante :

— Comme le montre la figure II. 11, un trou ne devant pas recevoir de broche ni de composant, sera constitué par un fil conducteur passant dans ce trou et soudé des deux côtés.

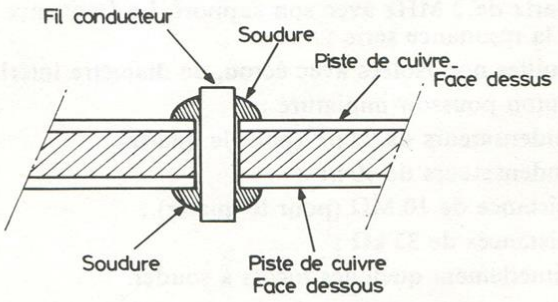


Fig. II. 11. « Trous métallisés » d'amateur

— Comme le montre la figure II. 12, un trou recevant un support de circuit intégré par exemple, *nécessitera une soudure du côté du support.* Ceci est possible seulement si les broches des supports sont suffisamment longues ; c'est pourquoi nous avons souvent utilisé des supports à wrapper, qu'on peut ainsi laisser à une certaine distance de la surface de la plaque de cuivre.

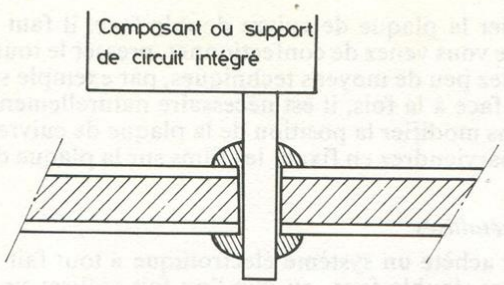


Fig. II. 12. Soudage des composants sur les deux faces

IV. 3. Matériel nécessaire à la réalisation de la maquette processeur

— 1 microprocesseur CDP 1802 CE boîtier plastique (RCA) ;
— 1 mémoire RAM 128 × 8 CDP 1823 CE boîtier plastique (RCA) ;
— 1 registre CD 4076 boîtier plastique (RCA) ;
— 1 support 24 broches à souder (pour la mémoire RAM) ;
— 1 support 40 broches à wrapper ;
— 1 support 24 broches à wrapper ;
— 1 support 16 broches à wrapper ;
— 1 quartz de 2 MHz avec son support. La fréquence indiquée doit être celle de la résonance série ;
— 2 douilles non isolées avec écrou, de diamètre intérieur 2 mm ;
— 1 bouton-poussoir miniature ;
— 2 condensateurs de 22 pF (pour le quartz) ;
— 3 condensateurs de 10 nF ;
— 1 résistance de 10 MΩ (pour le quartz) ;
— 7 résistances de 33 kΩ ;
— Eventuellement quelques picots à souder.

Remarque

Les valeurs des composants ne sont pas critiques. On pourra prendre par exemple des résistances 1/4 W, 10 %.

IV. 4. Réalisation pratique effective

Le lecteur dispose donc du circuit imprimé. Le diamètre de perçage des trous doit être le suivant :

— Ø 0,9 mm pour les trois supports à wrapper ;

— Ø 0,8 mm pour le support à souder, les condensateurs et les résistances, ainsi que les « trous métallisés » ;

— Ø 4,4 mm pour les douilles de 2 mm ;

— Ø 6,3 mm pour le bouton poussoir ;

— Ø 1,3 mm pour les trous devant recevoir un picot à souder.

Le schéma d'implantation des composants est donné à la figure II. 13, et l'explication des symboles utilisés figure II. 14.

Pour la réalisation, nous conseillons d'adopter la procédure suivante :

1. Souder les 3 « supports à wrapper » en n'oubliant pas de souder des deux côtés.

2. Souder le « support à souder » (on ne soude que d'un côté).

3. Réaliser les « trous métallisés » comme indiqué figure II.11, et souder les éventuels picots à souder (ceux-ci ne sont pas indispensables, mais permettent d'accéder sans connecteur aux broches Q, $\overline{EF_i}$, N_i, $\overline{DMA\ OUT}$ et SC 1 du microprocesseur).

4. Vérifier, en vous aidant du brochage des circuits et du schéma de câblage, que les connexions (spécialement adresses et données) entre les circuits intégrés sont effectives. On prendra un ohmmètre ou une sonde de sa convenance. Il est nécessaire également de vérifier qu'aucun court-circuit n'a été exécuté entre deux broches voisines du support d'un circuit intégré.

5. Souder le support du quartz.

6. Souder les composants passifs (résistances et condensateurs) en n'oubliant pas éventuellement de souder du côté de la face dessus.

7. Placer les douilles de 2 mm et les souder sur les pistes du circuit imprimé.

8. Câbler le bouton-poussoir. Vérifier, après avoir mis la maquette sous tension (sans avoir mis les circuits intégrés), que l'entrée $\overline{CLEAR}$ (broche 3 du microprocesseur) passe au niveau O lorsque le bouton-poussoir est activé.

9. Après avoir fait un ultime examen de l'état des soudures, on peut placer les circuits intégrés dans leur support.

Remarques

Dans l'état actuel des choses, il n'est pas possible de tester le fonctionnement de la maquette processeur, puisqu'aucun programme n'est encore disponible.

Les soudures doivent être exécutées avec un fer maniable et du fil à souder fin. Le travail demande un très grand soin et exige une bonne disponibilité d'esprit. La réussite de votre entreprise (si vous avez décidé de nous suivre) en dépend.

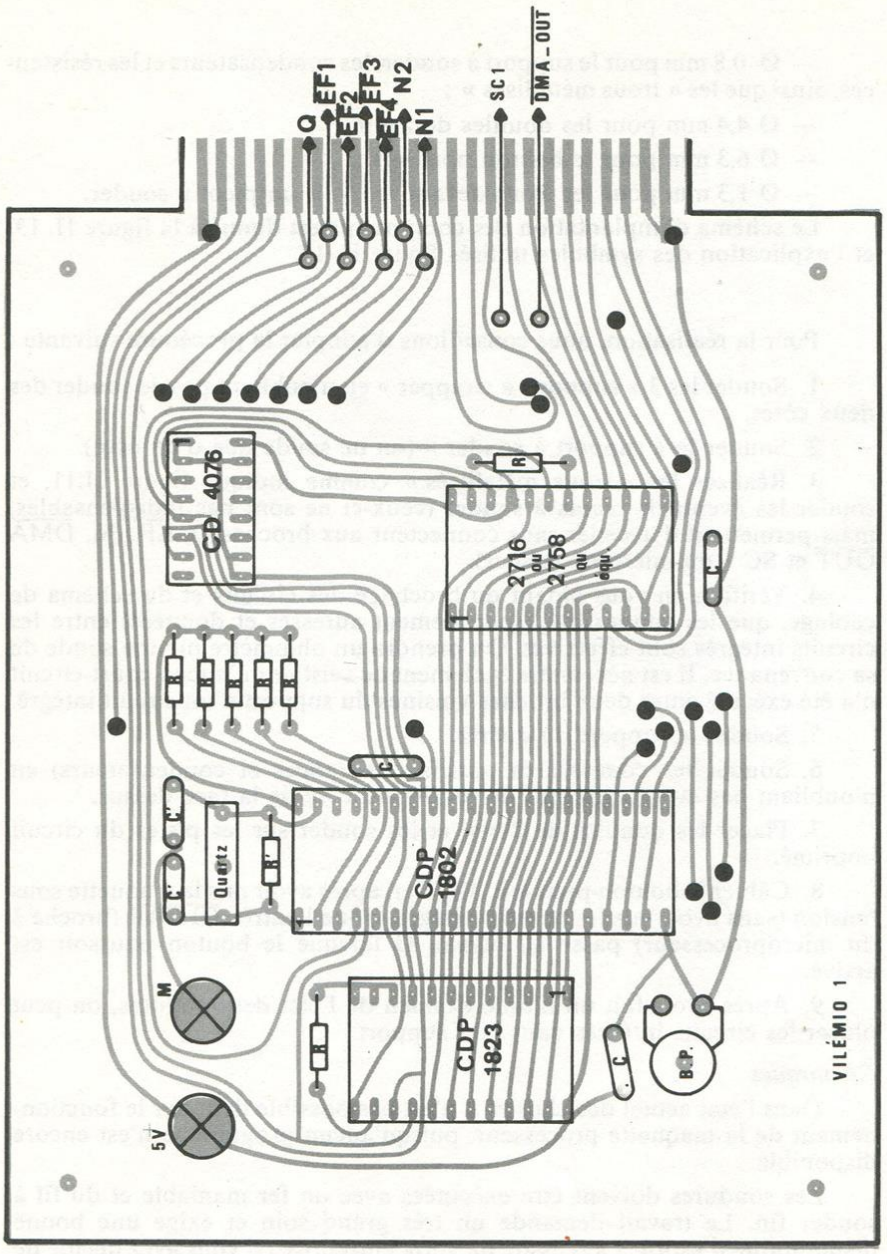


Fig. II. 13. Plan d'implantation des composants

Symboles utilisés

Douille de 2 mm

● « Trou métallisé »

○ Emplacement d'un picot à souder (facultatif)

$R = 33\ k\Omega$
$R' = 10\ M\Omega$
$C = 10\ nF$
$C' = 22\ pF$

Bouton-poussoir de remise à zéro

Fig. II. 14. Symboles utilisés sur la figure II-13

Chapitre 3

CONCEPTION ET RÉALISATION PRATIQUE DE L'ENSEMBLE « MAQUETTE MISE AU POINT - MAQUETTE CLAVIER-AFFICHEUR »

I. Conception de l'ensemble dit VILÉMIO 2/3

I. 1. Finalité du système

Si vous êtes un électronicien (ou un futur électronicien) connaissant déjà le principe de fonctionnement d'un microprocesseur, vous avez des idées générales sur la façon de l'utiliser. Vous désirez maintenant développer des applications. De quoi avez-vous besoin pour cela ?

Vous avez besoin d'un système qui contient une certaine quantité de mémoire RAM, pour écrire vos programmes. Compte tenu de notre expérience, nous considérons que *2 K octets de RAM* constituent une mesure correcte ; cette taille correspond d'ailleurs à la capacité d'une mémoire EPROM 2716 ou équivalente. Comme on ne peut mettre au point un programme de longueur 2 K octets en une journée, vous avez aussi besoin d'avoir la possibilité de pouvoir débrancher l'alimentation + 5 V de votre système, sans pour autant détruire le contenu de la RAM ; on dit alors qu'on dispose d'une *mémoire RAM permanente.* Au bout de longs jours ou de semaines épuisants, tant par l'angoisse ressentie parce que vous n'arriviez pas à résoudre immédiatement un problème, que par la joie éprouvée lorsqu'il était résolu, lorsque vous avez finalement mis au point votre programme, il est temps de le figer en EPROM ; vous avez donc également besoin d'un *programmeur d'EPROM.* Si, pour terminer, vous désirez vous constituer une bibliothèque de programmes, ou si pour une raison ou une autre vous désirez conserver vos programmes sur une bande magnétique, vous avez besoin de ce qu'on appelle couramment un *interface cassette.* C'est le rôle de la maquette « mise au point » d'offrir à l'amateur intéressé cet « outillage », propre à permettre l'élaboration, de A à Z, d'une application à microprocesseur.

I. 2. L'accès à la RAM de la maquette « mise au point » dite VILÉMIO 2

Quel moyen utiliser pour enregistrer dans la RAM la suite de mots de 8 bits qui constitue votre programme ?

Il y a essentiellement deux procédés :

a) Le microprocesseur CDP 1802 que nous utilisons possède une entrée $\overline{\text{DMA-IN}}$ qui, activée (c'est-à-dire mise au niveau 0), autorise un accès direct à la mémoire en vue de l'écrire. Comme nous l'avons déjà expliqué dans notre précédent ouvrage déjà cité, c'est le registre R (0) interne au microprocesseur qui adresse alors la mémoire ; par ailleurs le microprocesseur est « débranché » du bus des données, ce qui rend l'entrée de la RAM directement accessible au monde extérieur. Lors d'un cycle DMA, le mot présent sur le bus des données et issu du monde extérieur (8 clés par exemple, dans le cas le plus simple) est transféré dans la RAM à l'adresse contenue dans le registre R (0) ; en fin de cycle DMA, R (0) s'incrémente automatiquement, ce qui permet l'enregistrement du mot suivant au cours d'un nouveau cycle DMA, et ainsi de suite.

Ce mode de chargement spécial de la mémoire, appelé mode LOAD, est propre à ce microprocesseur, et présente l'avantage de ne nécessiter qu'un très petit nombre de composants.

b) L'autre procédé consiste à doter le système d'une mémoire *morte* contenant un programme (appelé *moniteur*) destiné précisément à enregistrer dans la RAM un programme (appelé programme *utilisateur* pour ne pas le confondre avec le programme moniteur).

C'est ce deuxième procédé que nous avons choisi, et ceci pour les raisons suivantes :

- il permet un accès plus souple à la mémoire, ce qui facilite la vie du programmeur lors de la mise au point de longs programmes ;
- comme de toute façon nous avions prévu de doter le système d'une mémoire morte contenant les routines destinées à la programmation des EPROM et à l'interface cassette, il aurait peut-être été « incohérent » de ne pas y inclure les routines destinées à l'accès à la RAM.

I. 3. L'occupation de l'espace mémoire

On demandera au lecteur de se reporter à la figure II. 2 du chapitre précédent. *Ce paragraphe définit l'espace mémoire dans le mode MISE AU POINT, c'est-à-dire lorsque les maquettes « processeur » et « mise au point » sont couplées entre elles.*

a) Comme nous venons de le dire, la maquette « mise au point » contient une mémoire morte contenant le programme moniteur ; il s'agit d'une EPROM 2716 (*) de capacité 2 K octets de chez INTEL. Comme nous n'utilisons que le premier K octets, *ce boîtier réside aux adresses de 0000 à 03FF ;* ainsi, lors de la mise sous tension ou lors de l'activation de l'entrée

(*) Si le lecteur ne peut se procurer cette EPROM *programmée*, les auteurs peuvent la fournir contre un chèque de **120 F** (programmation et port compris), **à l'ordre de Mr Villard** et adressé à ETSF, 2 à 12 rue de Bellevue, 75940 Paris cedex 19. Rappeler le titre de l'ouvrage avec la commande.

$\overline{\text{CLEAR}}$ du microprocesseur, c'est le programme moniteur qui « a la main », et le système est prêt à recevoir vos données. A noter que nous expliquons en détail le programme moniteur dans la 2e partie de cet ouvrage.

b) La RAM 2 K octets est divisée en deux groupes de 1 K octets notés RAM I et RAM II. Chaque groupe est constitué par deux boîtiers de capacité 1024 × 4. L'ensemble réside aux adresses *de 0400 à 0BFF.*

c) N'oublions pas que la maquette « mise au point » est nécessairement couplée à la maquette « processeur », et que celle-ci contient une RAM de capacité 128 octets résidant aux adresses de *0C00 à 0C7F.* Ainsi en réalité, la capacité totale en mémoire RAM disponible est de 2 K octets plus 128 octets.

d) La maquette processeur contient par ailleurs *éventuellement* une EPROM 2 K correspondant à une application donnée, et ceci par exemple lorsqu'on désire faire une copie de son contenu dans une autre EPROM. Lorsqu'on couple les maquettes « processeur » et « mise au point » entre elles, l'entrée $\overline{CE}$ de l'EPROM 2K est reliée à $\overline{A_{12}}$, ce qui définit son lieu d'habitation aux adresses de *1000 à 17FF.*

e) Pour disposer des entrées et sorties nécessaires à la commande des afficheurs, du clavier et de l'EPROM à programmer, nous avons choisi un boîtier 8255 de chez INTEL (voir plus loin) qui occupe les 4 lignes mémoire d'adresses *2000 à 2003.*

I. 4. La sauvegarde du contenu de la RAM

Lorsqu'on utilise des mémoires RAM CMOS, leur contenu est sauvegardé à condition de les alimenter sous au moins 2 V. La source de tension débite alors un courant tellement faible que nous n'avons pu le mesurer ; nous avons utilisé un ampèremètre donnant une déviation à pleine échelle de 100 divisions pour 1 μA. Comme l'aiguille n'a pas dévié, nous estimons que le courant débité était inférieur à 10 nA, courant lui-même inférieur au courant de fuite de l'accumulateur qui alimente la RAM.

La *figure III. 1a* exprime la première idée : lorsque l'alimentation 5 V disparaît, la diode D est passante et l'accumulateur alimente la RAM. Lorsque l'alimentation 5 V est placée, D est bloquée et D' est passante et la RAM est alimentée sous 5 V moins la tension de seuil de D' ; qu'on prenne pour D' du silicium ou du germanium, sa tension de seuil est de l'ordre de 0,7 V parce que les RAM que nous utilisons consomment un courant de plusieurs mA en cours de fonctionnement. Ainsi la RAM est alimentée sous environ 4,3 V. Or, les caractéristiques du constructeur indiquent que les entrées du boîtier RAM ne peuvent être soumises à des tensions supérieures à $V_{CC} + 0{,}7$ V, soit 5 V ; les lignes d'adresses pouvant précisément être à ce niveau, nous sommes *juste à la limite* des possibilités du boîtier RAM. Comme nous avons eu peur (peut-être à tort) nous avons pensé à remplacer la diode D' par un transistor T.

La *figure III. 1b* exprime cette idée : si la résistance R_B est bien choisie

(le courant I_B doit être suffisant), le transistor T est saturé quand l'alimentation 5 V est placée. La tension V_{CESat} étant de l'ordre de 0,2 V, la RAM est maintenant alimentée sous 4,8 V ce qui convient. Malheureusement, lorsque l'alimentation 5 V est supprimée, l'accumulateur 2,7 V alimente bien la RAM mais débite aussi dans la résistance R_B car la jonction Collecteur-Base de T devient passante, ce qui ne convient pas.

D'où l'amélioration finale présentée à la *figure III. 1c :* lorsque l'alimentation 5 V est absente, l'accumulateur 2,7 V débite seulement dans la

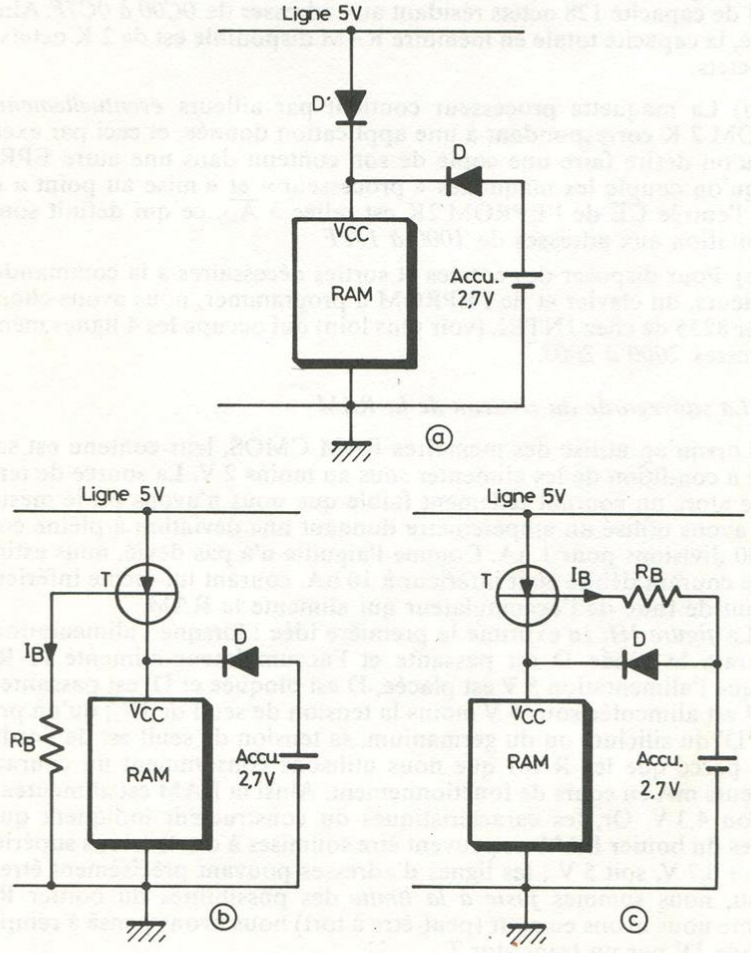


Fig. III. 1. Sauvegarde du contenu de la RAM

RAM, ce qui convient. Remarquez que lorsque l'alimentation 5 V est placée, le sens du courant I_B est tel que l'accumulateur *se charge*. Le courant I_B est donné par l'expression $I_B \frac{5-V_{EB}-2,7}{R_B} = \frac{1,6\ V}{R_B}$; compte tenu du courant de l'ordre de 30 mA absorbé par les 4 boîtiers RAM que nous utilisons, nous avons choisi $R_B = 820\,\Omega$ pour saturer correctement T.

Remarque importante

Si pour une raison ou une autre, on n'utilise pas d'accumulateur, celui-ci *doit* être *remplacé par un court-circuit*. Sinon, T n'est pas saturé et la RAM est alimentée sous une tension trop faible, ce qui peut l'endommager.

I. 5. La sélection des boîtiers

La figure III. 2 représente le plan de câblage de la logique de sélection des boîtiers mémoire (y compris le boîtier 8255). Donnons quelques mots d'explication :

a) Pour placer l'EPROM 2 K de la maquette processeur dans l'espace mémoire à partir de l'adresse 1000, et le boîtier 8255 de la maquette « mise au point » à partir de l'adresse 2000, il est nécessaire de disposer des bits d'adresse A_{12} et A_{13}. Ceux-ci sont générés par le registre de 2 bits CD 4013 déclenché par le signal TPA. $\overline{A_{12}}$ est relié au $\overline{CE}$ de l'EPROM 2K, et $\overline{A_{13}}$ au $\overline{CS}$ du boîtier 8255.

b) Les bits d'adresse A_{12}, A_{11} et A_{10} sont à l'entrée d'un démultiplexeur CD 4051 utilisé en décodeur.

Si $(A_{12}, A_{11}, A_{10}) = (0, 0, 0)$ la sortie notée (0) est au niveau 1.

Si $(A_{12}, A_{11}, A_{10}) = (0, 0, 1)$ la sortie notée (1) est au niveau 1.

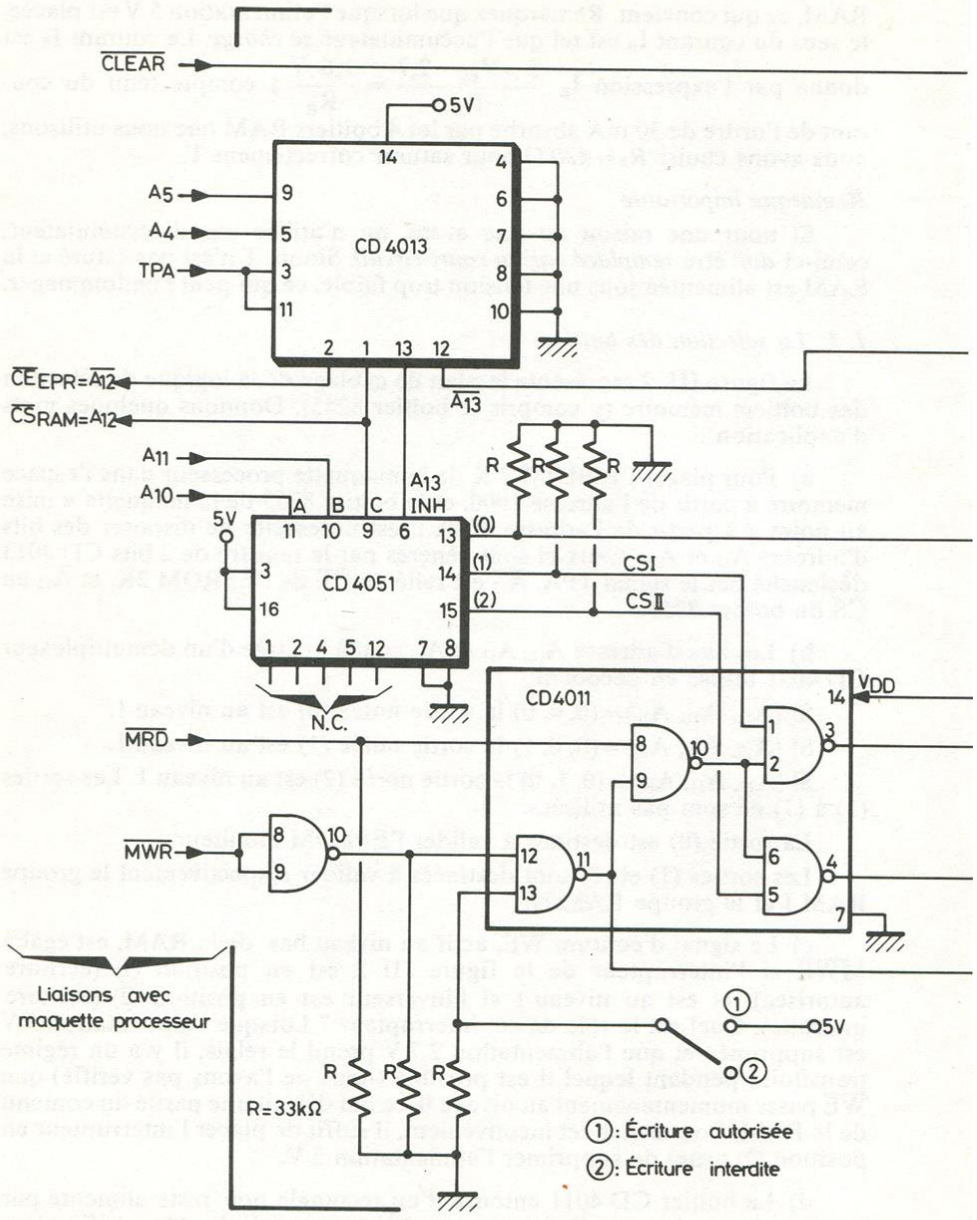
Si $(A_{12}, A_{11}, A_{10}) = (0, 1, 0)$ la sortie notée (2) est au niveau 1. Les sorties (3) à (7) ne sont pas utilisées.

La sortie (0) est destinée à valider l'EPROM moniteur.

Les sorties (1) et (2) sont destinées à valider respectivement le groupe RAM I et le groupe RAM II.

c) Le signal d'écriture $\overline{WE}$, actif au niveau bas de la RAM, est égal à $\overline{MWR}$ si l'interrupteur de la figure III. 2 est en position ① (écriture autorisée), et est au niveau 1 si l'inverseur est en position ② (écriture interdite). Quel est le rôle de cet interrupteur ? Lorsque l'alimentation 5 V est supprimée et que l'alimentation 2,7 V prend le relais, il y a un régime transitoire pendant lequel il est possible (nous ne l'avons pas vérifié) que $\overline{WE}$ passe momentanément au niveau 0, ce qui détruit une partie du contenu de la RAM. Pour éviter cet inconvénient, il suffit de placer l'interrupteur en position ② *avant* de supprimer l'alimentation 5 V.

d) Le boîtier CD 4011 entouré d'un rectangle noir reste alimenté par l'accumulateur lorsque l'alimentation 5 V est supprimée. On vérifie alors


CLEAR
5V
14
4
A5
9
6
A4
5
CD 4013
7
TPA
3
8
11
10
2 1 13 12
CE EPR=A12
CS RAM=A12
A13
A11
A10
A13 INH
A B C
R R R
5V
11 10 9 6 13
(0)
3
CD 4051
14
(1)
CSI
16
15
(2)
CSII
1 2 4 5 12 7 8
N.C.
CD4011
VDD
14
MRD
MWR
Liaisons avec maquette processeur
R R R
R=33kΩ
①
5V
②
①: Écriture autorisée
②: Écriture interdite

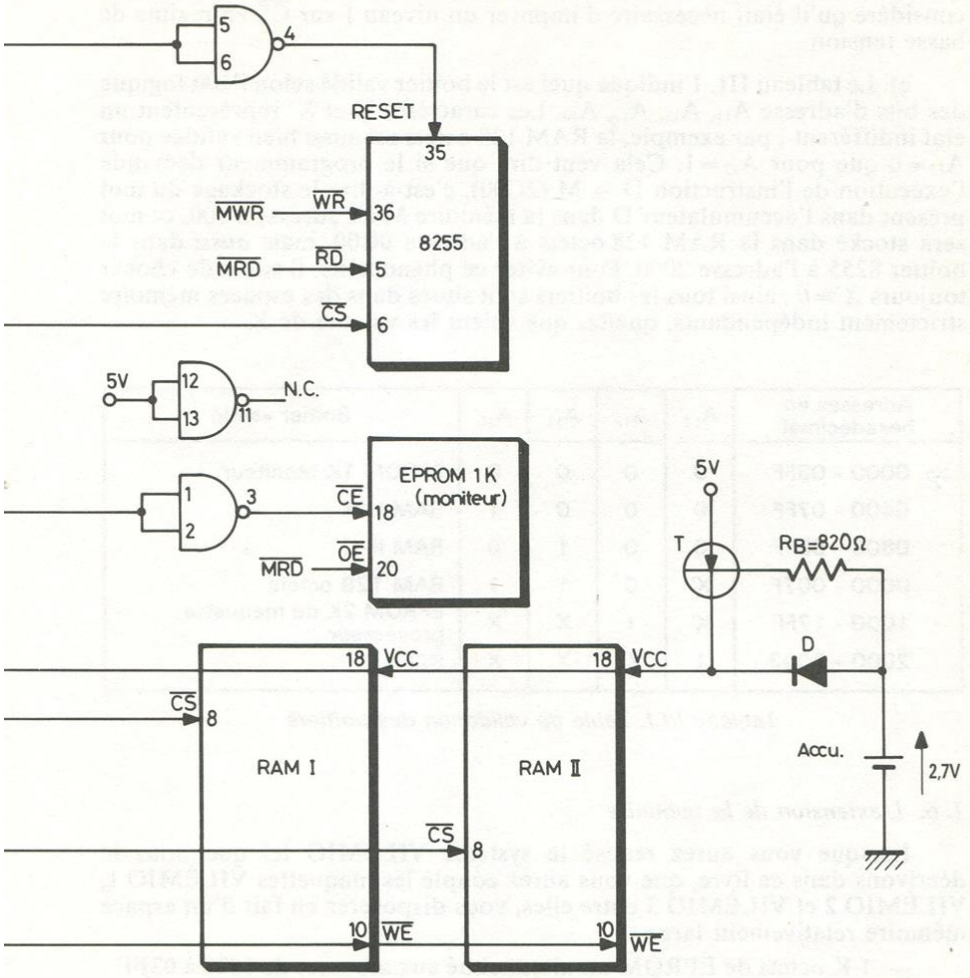


Fig. III. 2. Logique de sélection des boîtiers

que les entrées $\overline{CS}$ des boîtiers RAM sont au niveau 1. En effet, nous avons considéré qu'il était nécessaire d'imposer un niveau 1 sur $\overline{CS}$ en régime de basse tension.

e) Le tableau III. 1 indique quel est le boîtier validé selon l'état logique des bits d'adresse A_{13}, A_{12}, A_{11}, A_{10}. Les caractères X et X' représentent un état indifférent ; par exemple, la RAM 128 octets est aussi bien validée pour $A_{13}=0$ que pour $A_{13}=1$. Cela veut dire que si le programmeur demande l'exécution de l'instruction D → M (2C00), c'est-à-dire le stockage du mot présent dans l'accumulateur D dans la mémoire M à l'adresse 2C00, ce mot sera stocké dans la RAM 128 octets à l'adresse 0C00, mais *aussi* dans le boîtier 8255 à l'adresse 2000. Pour éviter ce phénomène, il suffit de choisir toujours *X'=0 ;* ainsi tous les boîtiers sont situés dans des espaces mémoire strictement indépendants, quelles que soient les valeurs de X.

Adresses en hexadécimal	A_{13}	A_{12}	A_{11}	A_{10}	Boîtier validé
0000 - 03FF	0	0	0	0	EPROM 1K Moniteur
0400 - 07FF	0	0	0	1	RAM I 1K
0800 - 0BFF	0	0	1	0	RAM II 1K
0C00 - 0C7F	X'	0	1	1	RAM 128 octets
1000 - 17FF	X'	1	X	X	EPROM 2K de maquette processeur
2000 - 2003	1	X'	X	X	8255

Tableau III.1. Table de validation des boîtiers

I. 6. L'extension de la mémoire

Lorsque vous aurez réalisé le système VILÉMIO tel que nous le décrivons dans ce livre, que vous aurez couplé les maquettes VILÉMIO 1, VILÉMIO 2 et VILÉMIO 3 entre elles, vous disposerez en fait d'un espace mémoire relativement large :

— 1 K octets de EPROM moniteur situé aux adresses de 0000 à 03FF ;

— 2 K octets de RAM situés aux adresses de 0400 à 0BFF ;

— 128 octets de RAM situés aux adresses de 0C00 à 0C7F ;

— 2 K octets de EPROM situé aux adresses de 1000 à 17FF. Cette dernière EPROM est placée sur la maquette processeur et contient un programme (ou des sous-programmes ou des données) que vous avez vous-même développé. Rappelons que cette EPROM (qu'il ne faut pas confondre avec l'EPROM moniteur) n'est pas nécessaire au fonctionnement de l'ensemble du système VILÉMIO.

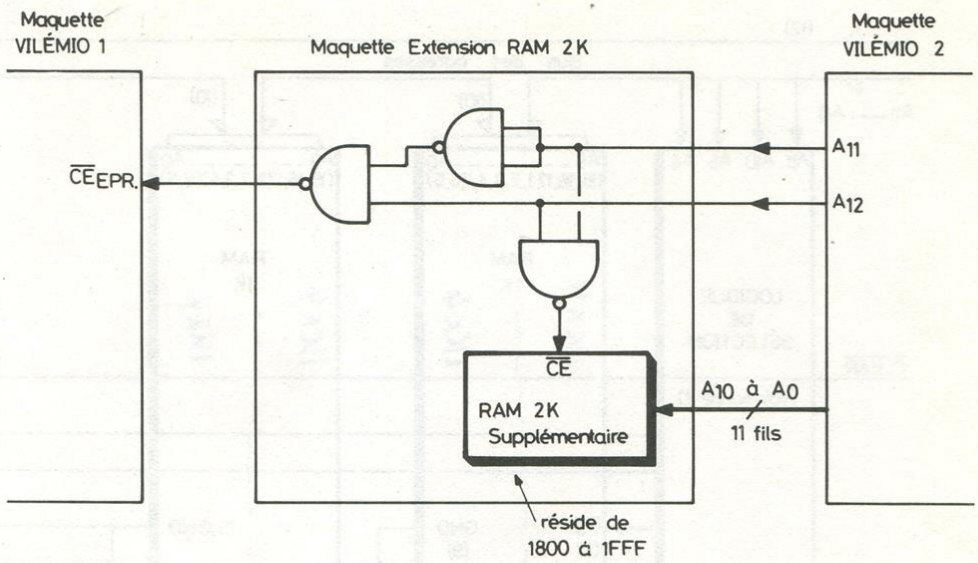


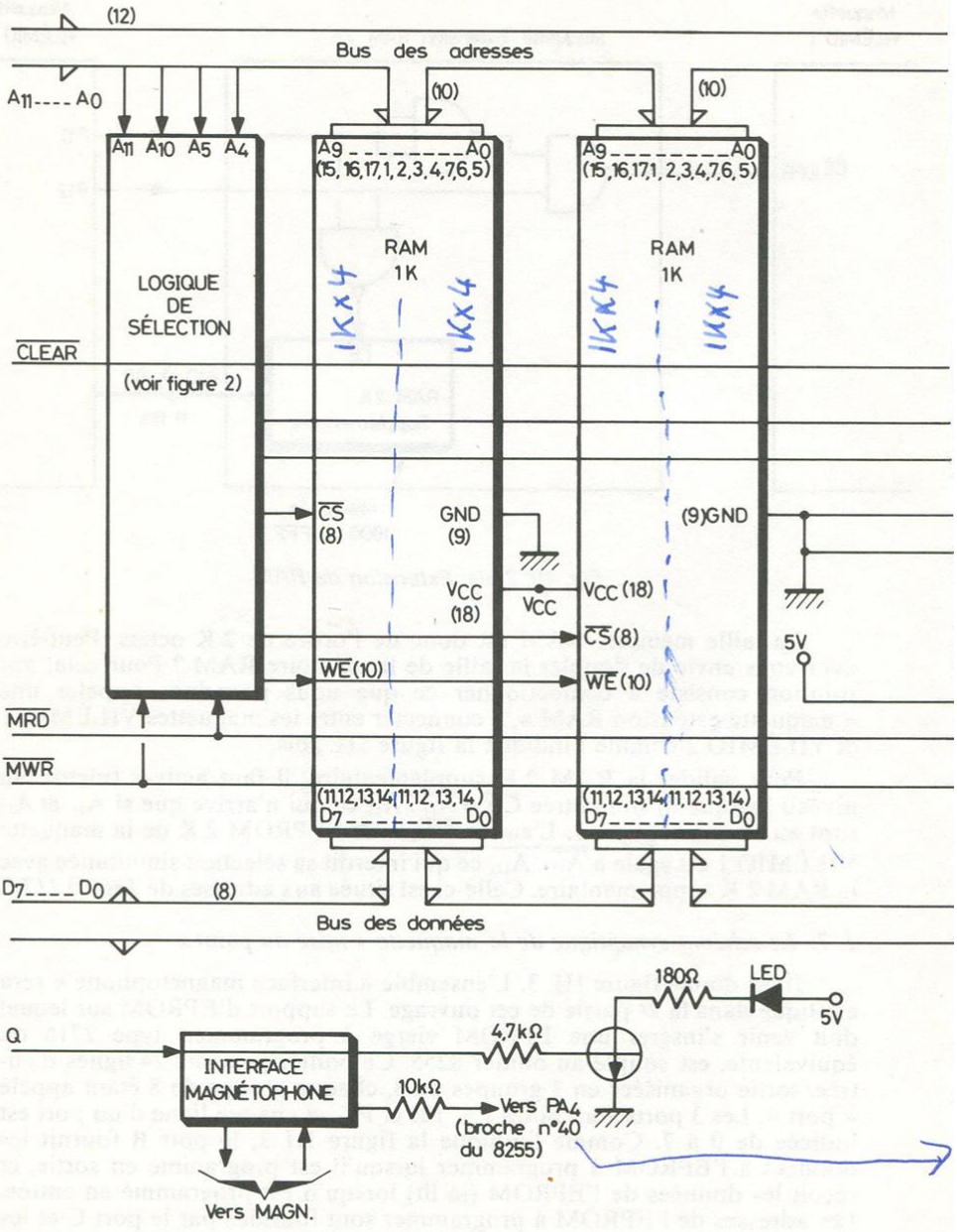
Fig. III. 2 bis. Extension de RAM

La taille mémoire RAM est donc de l'ordre de 2 K octets. Peut-être avez-vous envie de doubler la taille de la mémoire RAM ? Pour cela, *une* solution consiste à confectionner ce que nous pourrions appeler une « maquette extension RAM », à connecter entre les maquettes VILÉMIO 1 et VILÉMIO 2 comme l'indique la figure III. 2bis.

Pour valider la RAM 2 K supplémentaire, il faut activer (mettre au niveau logique 0) son entrée $\overline{CE} = \overline{A_{11}} \cdot A_{12}$ ce qui n'arrive que si A_{11} *et* A_{12} sont au niveau logique 1. L'entrée $\overline{CE}_{EPR}$ de l'EPROM 2 K de la maquette VILÉMIO 1 est égale à $\overline{A_{11}} \cdot A_{12}$, ce qui interdit sa sélection simultanée avec la RAM 2 K supplémentaire. Celle-ci est située aux adresses de *1800 à 1FFF.*

I. 7. Le schéma synoptique de la maquette « mise au point »

Il est donné figure III. 3. L'ensemble « interface magnétophone » sera expliqué dans la 2e partie de cet ouvrage. Le support d'EPROM sur lequel doit venir s'insérer une EPROM vierge à programmer, type 2716 ou équivalente, est couplé au boîtier 8255. Ce boîtier présente 24 lignes d'entrée/sortie organisées en 3 groupes de 8, chaque groupe de 8 étant appelé « port ». Les 3 ports sont notés PA, PB et PC, et chaque ligne d'un port est indicée de 0 à 7. Comme l'indique la figure III. 3, le port B fournit les données à l'EPROM à programmer lorsqu'il est programmé en sortie, et reçoit les données de l'EPROM (la lit) lorsqu'il est programmé en entrée. Les adresses de l'EPROM à programmer sont fournies par le port C et les


(12)
Bus des adresses
A11---- A0
(10)
(10)
A11 A10 A5 A4
A9 ---------A0
(15,16,17,1,2,3,4,7,6,5)
A9 ---------A0
(15,16,17,1,2,3,4,7,6,5)
RAM
1K
RAM
1K
1Kx4
1Kx4
1Kx4
1Kx4
LOGIQUE
DE
SÉLECTION
CLEAR
(voir figure 2)
CS
(8)
GND
(9)
(9)GND
VCC
(18)
VCC
VCC (18)
CS (8)
5V
WE (10)
WE (10)
MRD
MWR
(11,12,13,14,11,12,13,14)
D7 ---------D0
(11,12,13,14,11,12,13,14)
D7 ---------D0
D7---- D0
(8)
Bus des données
180Ω
LED
5V
Q
INTERFACE
MAGNÉTOPHONE
4,7kΩ
10kΩ
Vers PA4
(broche n°40
du 8255)
Vers MAGN.

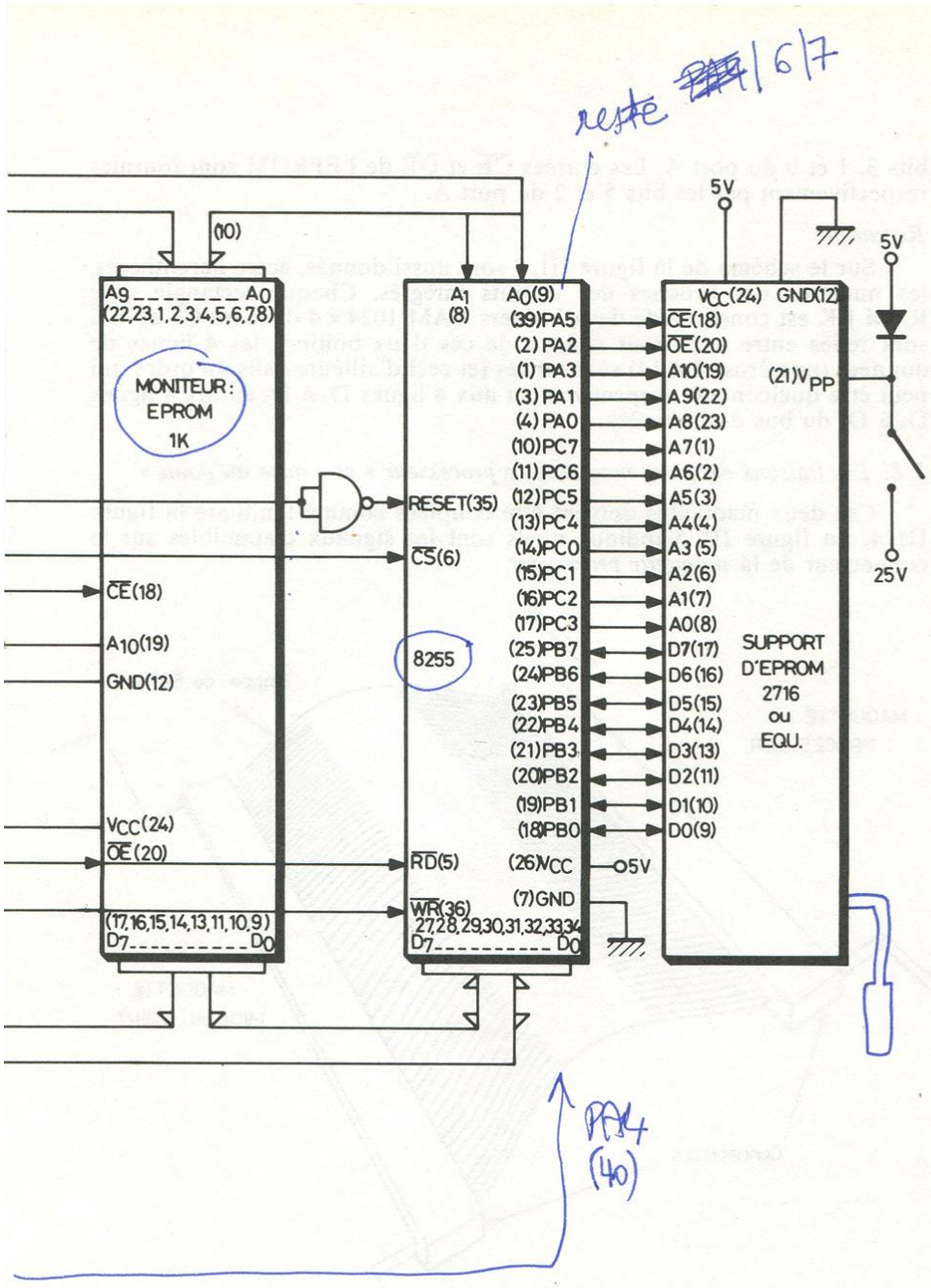


Fig. III. 3. Schéma synoptique de la maquette « mise au point »

bits 3, 1 et 0 du port A. Les entrées $\overline{CE}$ et $\overline{OE}$ de l'EPROM sont fournies respectivement par les bits 5 et 2 du port A.

Remarque

Sur le schéma de la figure III. 3 sont aussi donnés, entre parenthèses, les numéros des broches des circuits intégrés. Chaque rectangle noté RAM 1 K est constitué de deux boîtiers RAM 1024×4 dont les $\overline{CS}$ et $\overline{WE}$ sont reliés entre eux. Pour chacun de ces deux boîtiers, les 4 lignes de données (numéros 11 à 14) sont reliées (et ceci d'ailleurs dans un ordre qui peut être quelconque) respectivement aux 4 lignes D_7 à D_4 et aux 4 lignes D_3 à D_0 du bus des données.

I. 8. Les liaisons entre les maquettes « processeur » et « mise au point »

Ces deux maquettes doivent être couplées comme l'indique la figure III. 4. La figure III. 5 indique quels sont les signaux disponibles sur le connecteur de la *maquette processeur.*

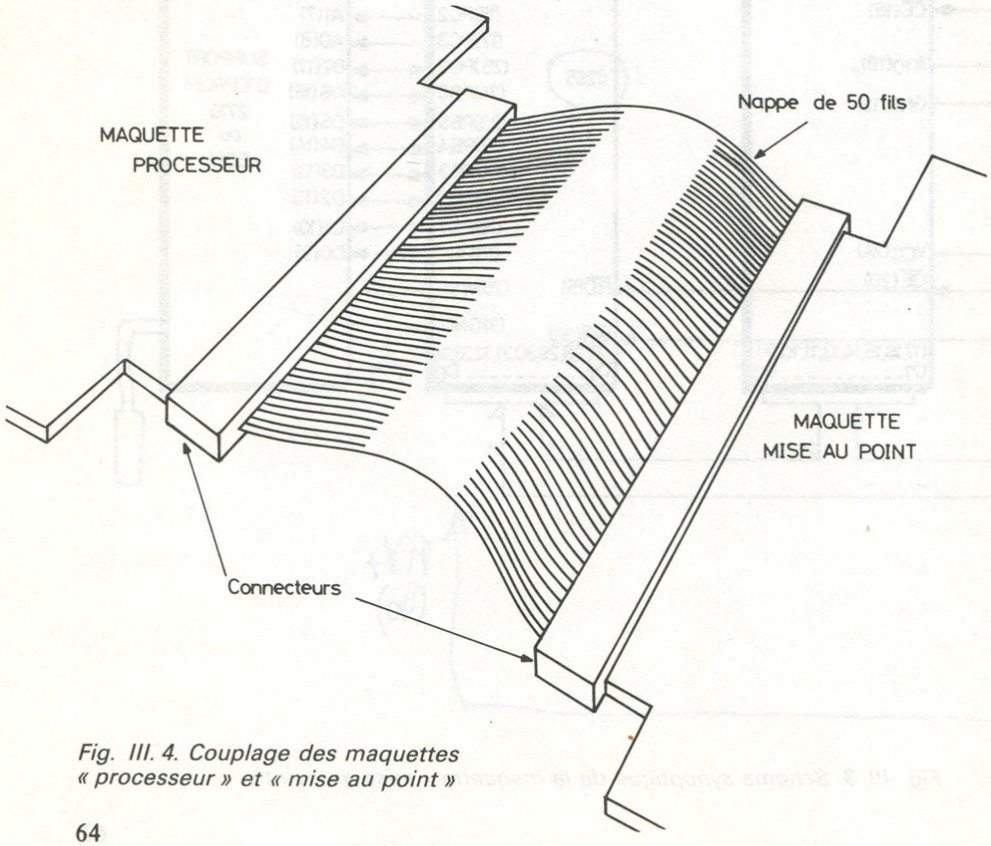
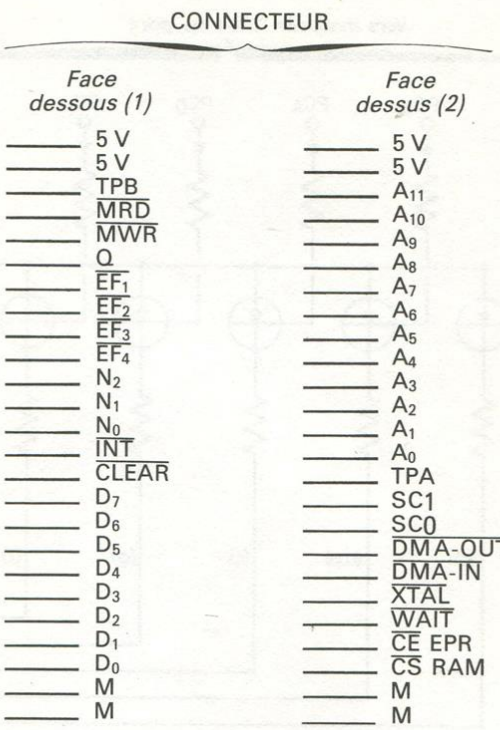


Fig. III. 4. Couplage des maquettes « processeur » et « mise au point »



(1) Face dessous pour la maquette « processeur » ; mais face *dessus* pour la maquette « mise au point ».

(2) Face dessus pour la maquette « processeur » ; mais face *dessous* pour la maquette « mise au point ».

Fig. III. 5. Distribution des signaux sur le connecteur

Pour notre part nous avons utilisé des connecteurs permettant une liaison par une nappe de 50 fils (cela évite d'avoir à faire des soudures, car ces connecteurs sont tels qu'une pression suffit à établir les contacts entre eux et les fils de la nappe). Il est à noter que le fait que les connecteurs soient vis-à-vis (voyez la figure III. 4) entraîne l'effet suivant : *les signaux de la face dessous de la maquette processeur se retrouvent sur la face dessus de la maquette mise au point, et réciproquement.* Cette remarque est fondamentale pour le lecteur désireux d'utiliser un autre mode de connexion, car il lui faudra alors faire attention à ce que les liaisons entre les deux maquettes soient telles qu'à tout point du connecteur de la maquette processeur corresponde le point du connecteur de la maquette mise au point sur la *face opposée.*

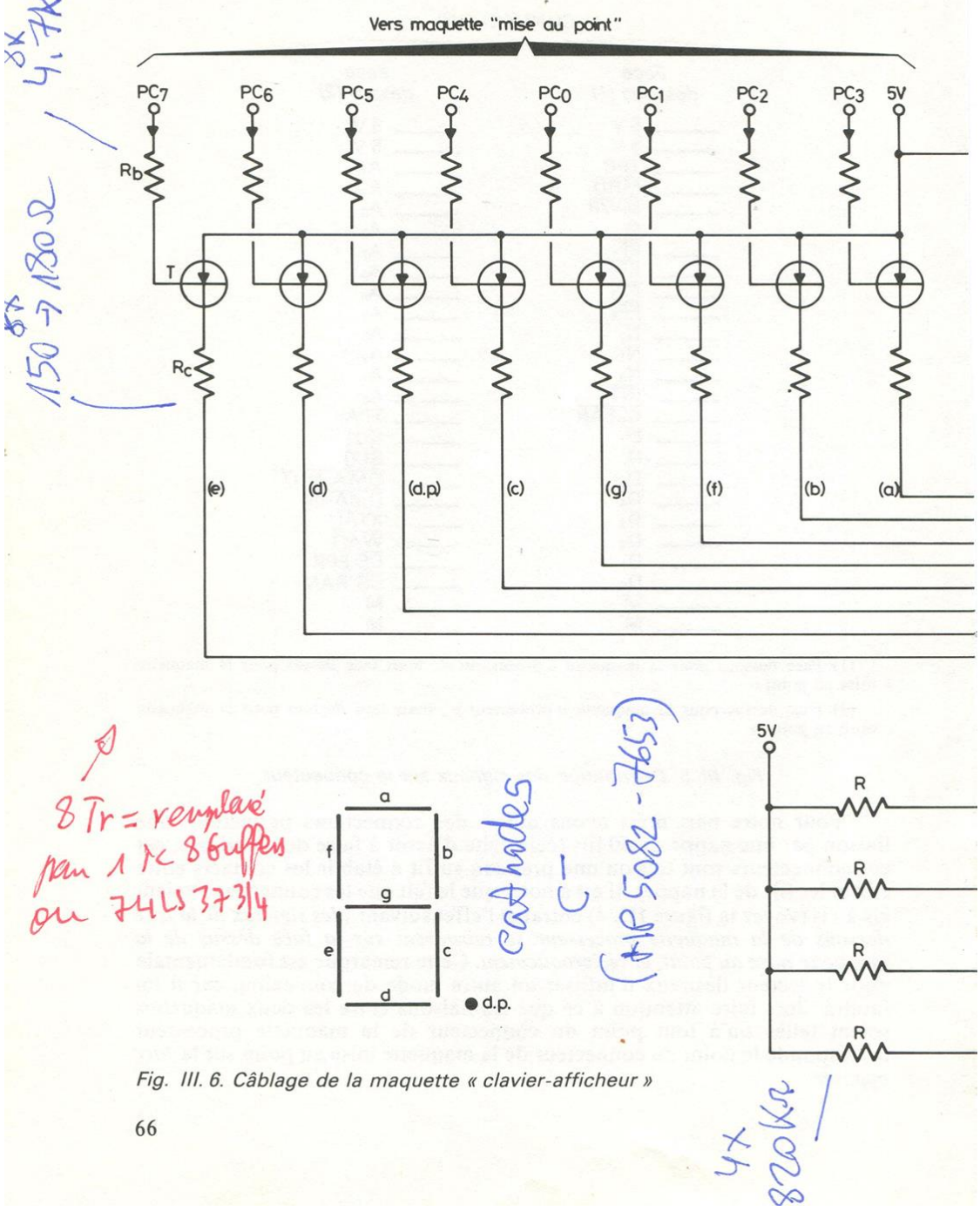
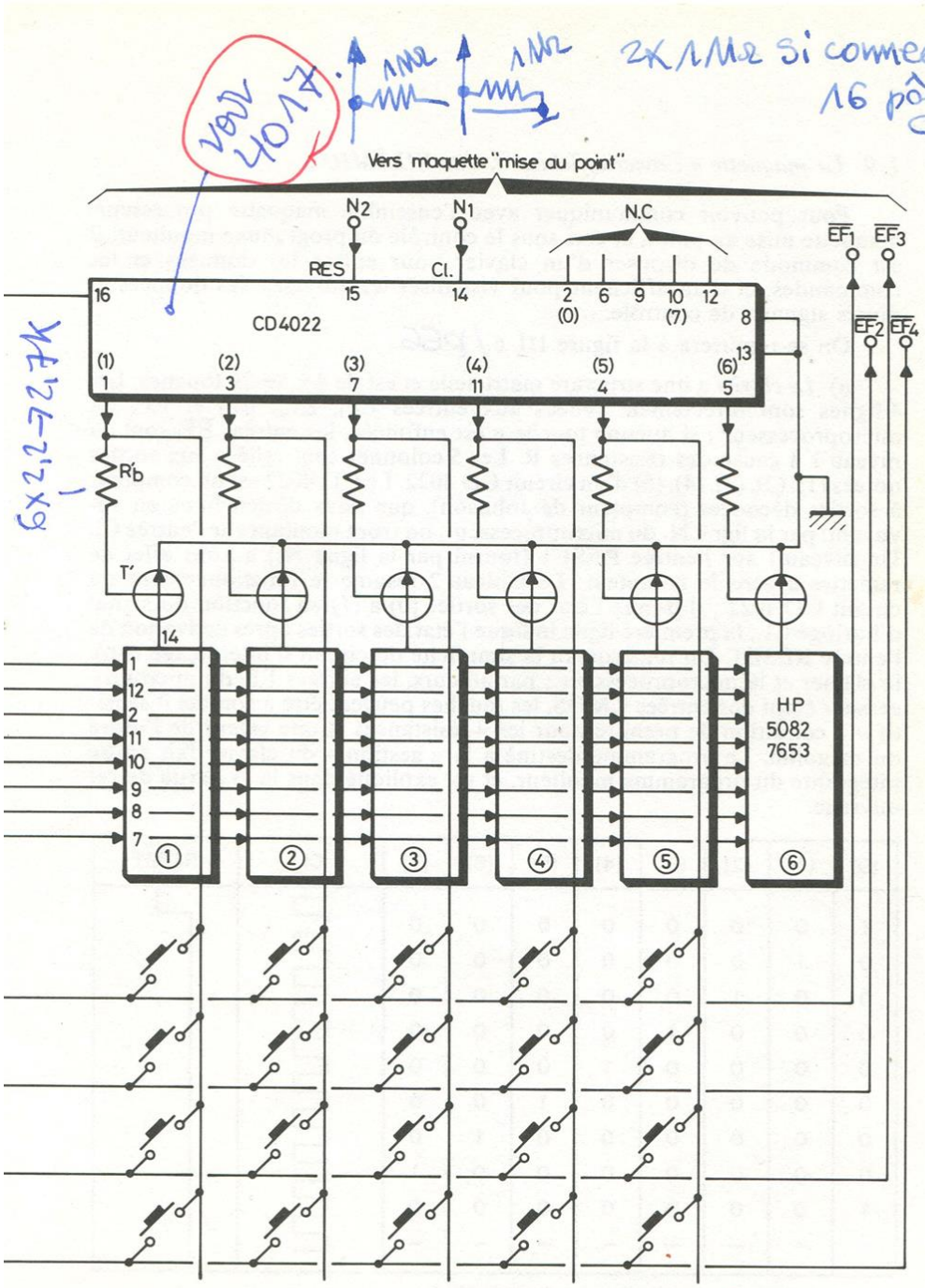


Fig. III. 6. Câblage de la maquette « clavier-afficheur »



4022/4017 Idem mais cablage faux.

T' = 1 transistor de 6 IR de PW voir Walman.

I. 9. La maquette « clavier-afficheur », dite VILÉMIO 3

Pour pouvoir communiquer avec l'ensemble maquette processeur-maquette mise au point, et ceci sous le contrôle du programme moniteur, il est commode de disposer d'un clavier pour entrer les données et les commandes, et d'un afficheur pour visualiser les adresses, les données et divers signaux de contrôle.

On se reportera à la figure III. 6./ p66

a) *Le clavier* a une structure matricielle et est de $4 \times 5 = 20$ touches. Les 4 lignes sont directement reliées aux entrées $\overline{EF_1}$, $\overline{EF_2}$, $\overline{EF_3}$ et $\overline{EF_4}$ du microprocesseur ; si aucune touche n'est enfoncée, les entrées $\overline{EF_i}$ sont au niveau 1 à cause des résistances R. Les 5 colonnes sont reliées aux sorties notées (1), (2), (3), (4), (5) d'un circuit CD 4022. Le CD 4022 est un compteur à sorties décodées (compteur de Johnson), que nous déclenchons en envoyant, par la ligne N_1 du microprocesseur, un front montant sur l'entrée Cl. Un niveau 1 sur l'entrée RESET (fourni par la ligne N_2) a pour effet de remettre à zéro le compteur. Le tableau 2 résume le fonctionnement du circuit CD 4022 ; il donne l'état des sorties (0) à (7) en fonction du signal d'horloge Cl ; la première ligne indique l'état des sorties après activation de l'entrée RESET. On remarquera la simplicité du circuit d'interfaçage entre le clavier et le microprocesseur ; par ailleurs, les entrées $\overline{EF_i}$ du microprocesseur étant des entrées CMOS, les touches peuvent être à contact « sensitif » à condition de prendre pour les 4 résistances R une valeur de l'ordre du mégohm. Le programme destiné à la « gestion » du clavier fait partie intégrante du programme moniteur, et est expliqué dans la 2e partie de cet ouvrage.

(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	Cl.	RESET
1	0	0	0	0	0	0	0		
0	1	0	0	0	0	0	0		
0	0	1	0	0	0	0	0		
0	0	0	1	0	0	0	0		
0	0	0	0	1	0	0	0		
0	0	0	0	0	1	0	0		
0	0	0	0	0	0	1	0		
0	0	0	0	0	0	0	1		
1	0	0	0	0	0	0	0		
–	–	–	–	–	–	–	–		

Tableau III.2.

b) *Les afficheurs* sont du type 7 segments plus un point à droite, à cathode commune, et sont au nombre de 6. Les afficheurs notés ① à ④ sont destinés essentiellement à l'affichage des adresses, et ceux notés ⑤ et ⑥ aux données. Les anodes des segments plus le point sont commandées individuellement par les 8 sorties PC_7 à PC_0 du boîtier 8255 de la maquette mise au point. Les 8 transistors PNP notés T servent à fournir un courant suffisant aux anodes. A noter que pour *allumer un segment, il faut envoyer un niveau bas* sur l'entrée PC correspondante. C'est la technique du multiplexage qui est employée ; par conséquent, on doit être en mesure de valider un seul des 6 afficheurs à la fois : c'est le rôle des 6 transistors NPN notés T' de laisser un libre passage ou non au courant circulant dans les cathodes communes des afficheurs. Ces 6 transistors T' sont commandés par les sorties 1 à 6 du boîtier CD 4022 ; ainsi, lorsque ce compteur tourne, les afficheurs sont validés un par un, successivement de gauche à droite. Le programme de « gestion » des afficheurs fait partie du programme moniteur.

c) Choix des résistances

Il semble que la valeur optimale du courant I_F devant circuler dans un segment est de 20 mA ; en effet, un courant plus important ne donne pas une luminosité nettement meilleure. La chute de tension V_F aux bornes d'un segment est alors de l'ordre de 1,8 V. La tension de saturation de chaque transistor étant environ de 0,2 V, la résistance R_C doit être choisie égale à $R_C = \frac{5 - 1{,}8 - 0{,}4}{20}$ soit $R_C = 140\ \Omega$. On prendra 150 Ω.

Les résistances R_b et R'_b doivent permettre la saturation de T et T' ; on prendra quelques milliers d'ohms (par exemple 2,2 kΩ).

I. 10. Les liaisons entre les maquettes « mise au point » et « clavier-afficheur »

Elles sont à effectuer comme l'indique la figure III. 7, avec des cordons

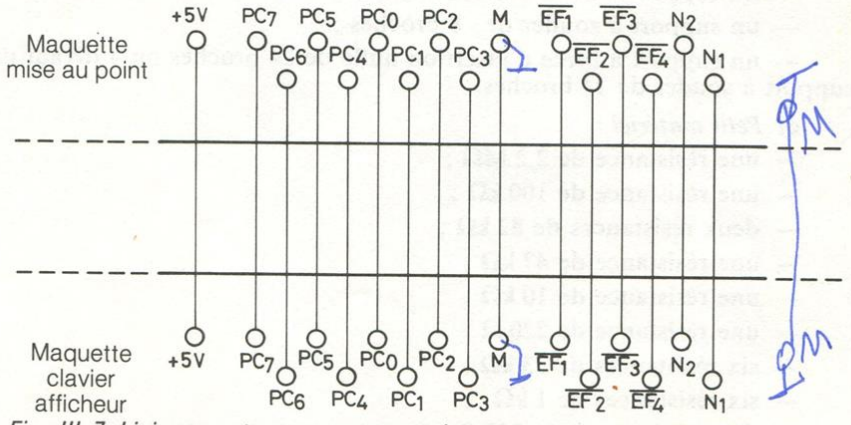


Fig. III. 7. Liaisons entre maquettes « mise au point » et « clavier-afficheur »

ou des fils soudés sur les picots correspondants. Si on n'utilise pas de fils soudés, on peut déplacer la maquette « clavier-afficheur » et la rendre partie intégrante d'une maquette application. De toute manière, la maquette « clavier-afficheur » couplée à la maquette « mise au point » peut cependant être considérée (et utilisée) telle quelle comme une maquette application ou un périphérique.

II. Réalisation pratique de l'ensemble VILÉMIO 2/3

II. 1. Matériel nécessaire à la réalisation de la maquette « mise au point »

a) *Circuits intégrés :*

— une EPROM 2716 (Intel) contenant le programme moniteur ;

— 4 mémoires RAM 1024 × 4 MWS 5114 (RCA) ou TC 5514 P 450 ns (TOSHIBA) ;

— un démultiplexeur CD 4051 (RCA) ;

— une double bascule D CD 4013 (RCA) ;

— trois boîtiers NAND CD 4011 (RCA) ;

— un double ampli op TL 072 CP (TEXAS) ou équivalent ;

— un coupleur 8255 (Intel).

b) *Supports des circuits intégrés :*

— un support à wrapper de 40 broches ;

— un support à wrapper de 24 broches ;

— quatre supports à wrapper de 18 broches ;

— un support à wrapper de 16 broches ;

— trois supports à wrapper de 14 broches ;

— un support à souder de 14 broches ;

— un support à souder de 8 broches ;

— un support à force d'insertion nulle de 24 broches ou à défaut un support à souder de 24 broches.

c) *Petit matériel :*

— une résistance de 2,2 MΩ ;

— une résistance de 100 kΩ ;

— deux résistances de 82 kΩ ;

— une résistance de 47 kΩ ;

— une résistance de 10 kΩ ;

— une résistance de 220 Ω ;

— six résistances de 33 kΩ ;

— six résistances de 1 kΩ ;

— deux résistances de 330 Ω ;

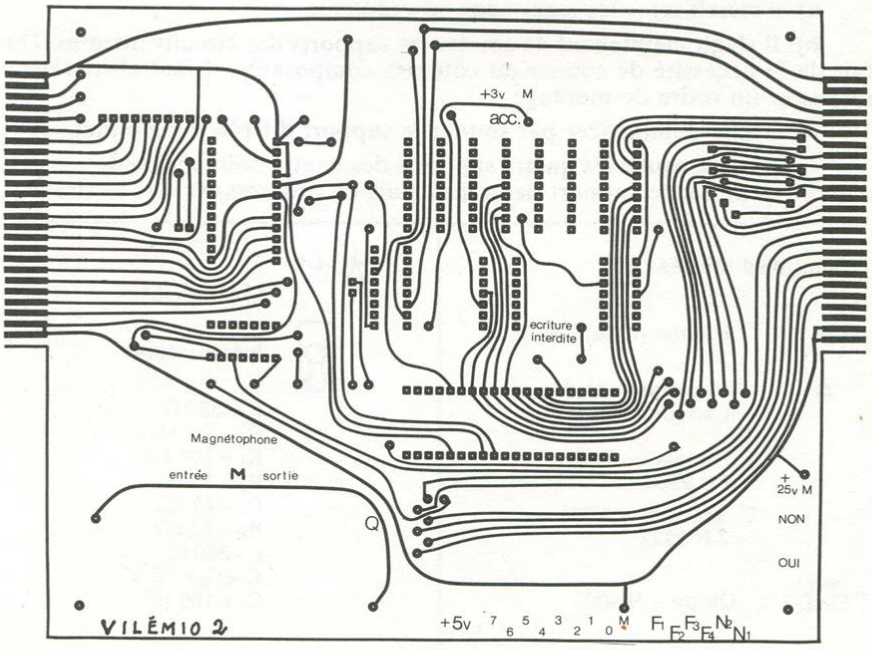
— une résistance de 820 Ω ;
— cinq condensateurs de 10 nF ;
— trois condensateurs de 100 nF ;
— un transistor 2 N 2904 (PNP) ;
— un transistor 2 N 2222 (NPN) ;
— quatre diodes 1 N 4002 ;
— une LED TIL 209.

d) *Connectique :*

— deux interrupteurs miniatures ;
— 25 picots à souder ;
— deux connecteurs 2 × 25 broches double face au pas de 2,54 mm, avec nappe de 50 fils (pour la liaison avec la maquette processeur).

II. 2. Réalisation pratique de la maquette « mise au point »

Le dessin du circuit imprimé est donné figures III. 8 et III. 9. La figure III. 8 représente le dessin du circuit imprimé vu côté composants, c'est-



(Voir pages 80-81 et 144-145 les figures III.8 et III.9 à l'échelle 1).

à-dire le dessin du circuit que vous voyez, la maquette étant placée normalement sur une table. Il s'agit de la face dessus.

La figure III. 9 représente le dessin du circuit imprimé tel qu'on le voit, la maquette étant posée retournée sur une table. Il s'agit de la face dessous.

Le diamètre des trous de perçage doit être le suivant :

— Ø 0,9 mm pour les trous devant contenir les supports à wrapper et pour le support à force d'insertion nulle ;

— Ø 0,8 mm pour les trous devant contenir les supports à souder, pour les condensateurs, les résistances, et les « trous métallisés ». Même diamètre pour les transistors ;

— Ø 1,1 mm pour les diodes 1 N 4002 ;

— Ø 1,3 mm pour les trous devant recevoir un picot à souder ;

— Ø 6,3 mm pour les interrupteurs miniatures.

Pour la réalisation pratique, on se reportera au schéma d'implantation des composants donné en figure III. 10. La figure III. 11 donne l'explication des symboles utilisés et la valeur numérique des composants passifs. Pour une réussite garantie, il est indispensable d'adopter la procédure de construction suivante :

a) « métalliser » les trous sans en oublier.

b) Il s'agit maintenant de souder les supports des circuits intégrés. Du fait de la nécessité de souder du côté des composants, il faut absolument respecter un ordre de montage :

— il faut commencer par souder le support d'EPROM moniteur ;

— puis on soude les quatre supports des quatre boîtiers RAM, mais en commençant par le support de *gauche* puis en progressant vers la *droite* ;

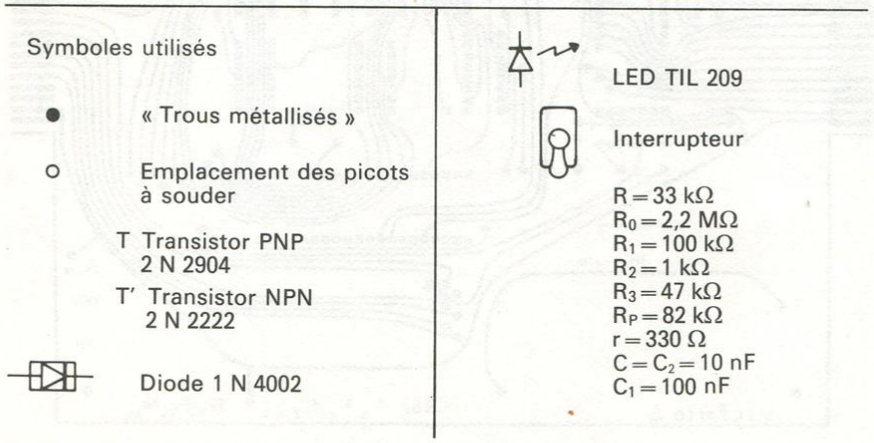


Fig. III.11 Symboles utilisés sur la figure III.10.

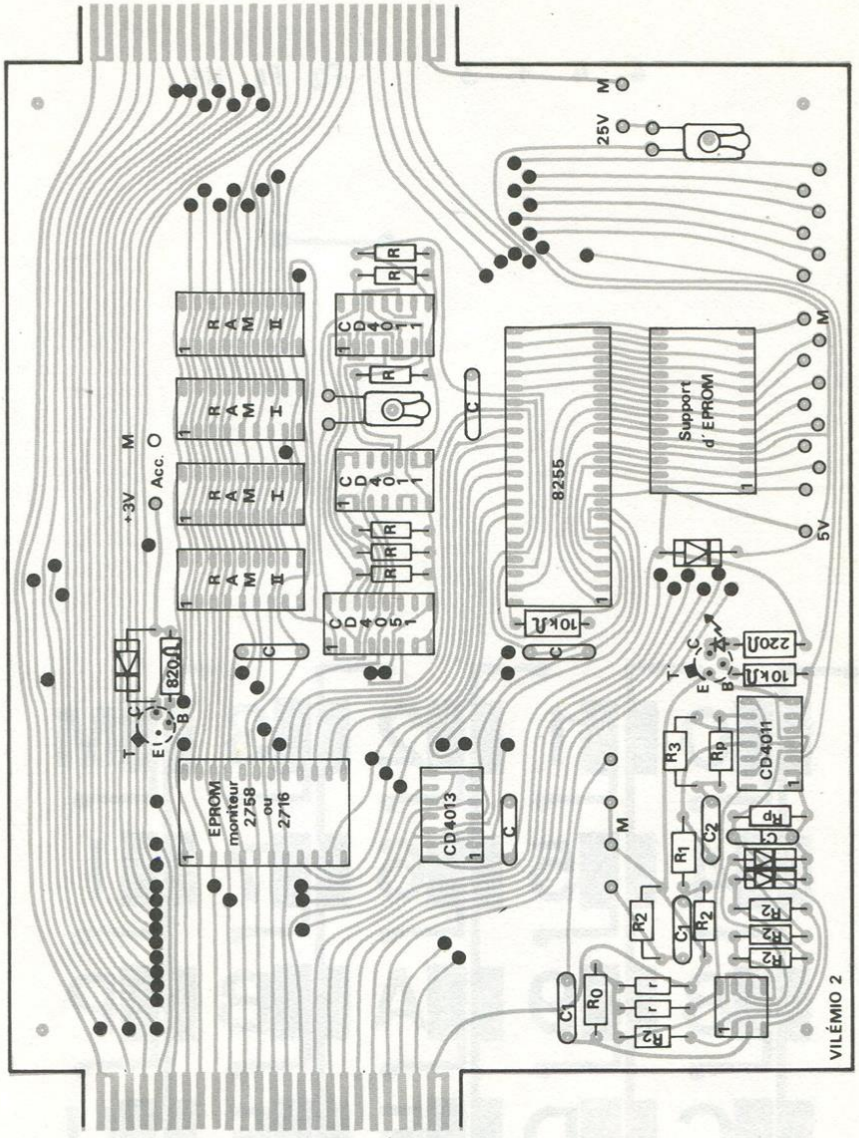


Fig. III. 10. Implantation des composants

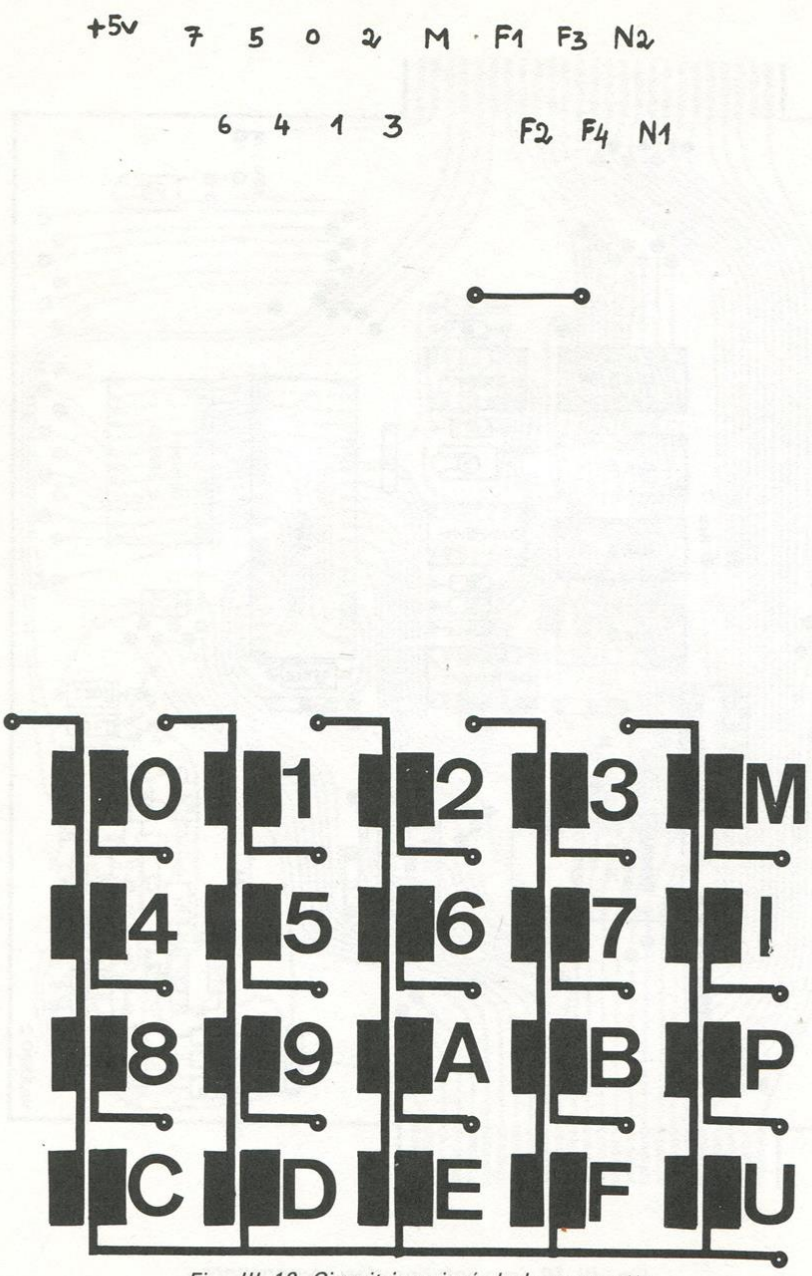


Fig. III. 12. Circuit imprimé de la maquette « clavier-afficheur » face dessus

VILÉMIO3

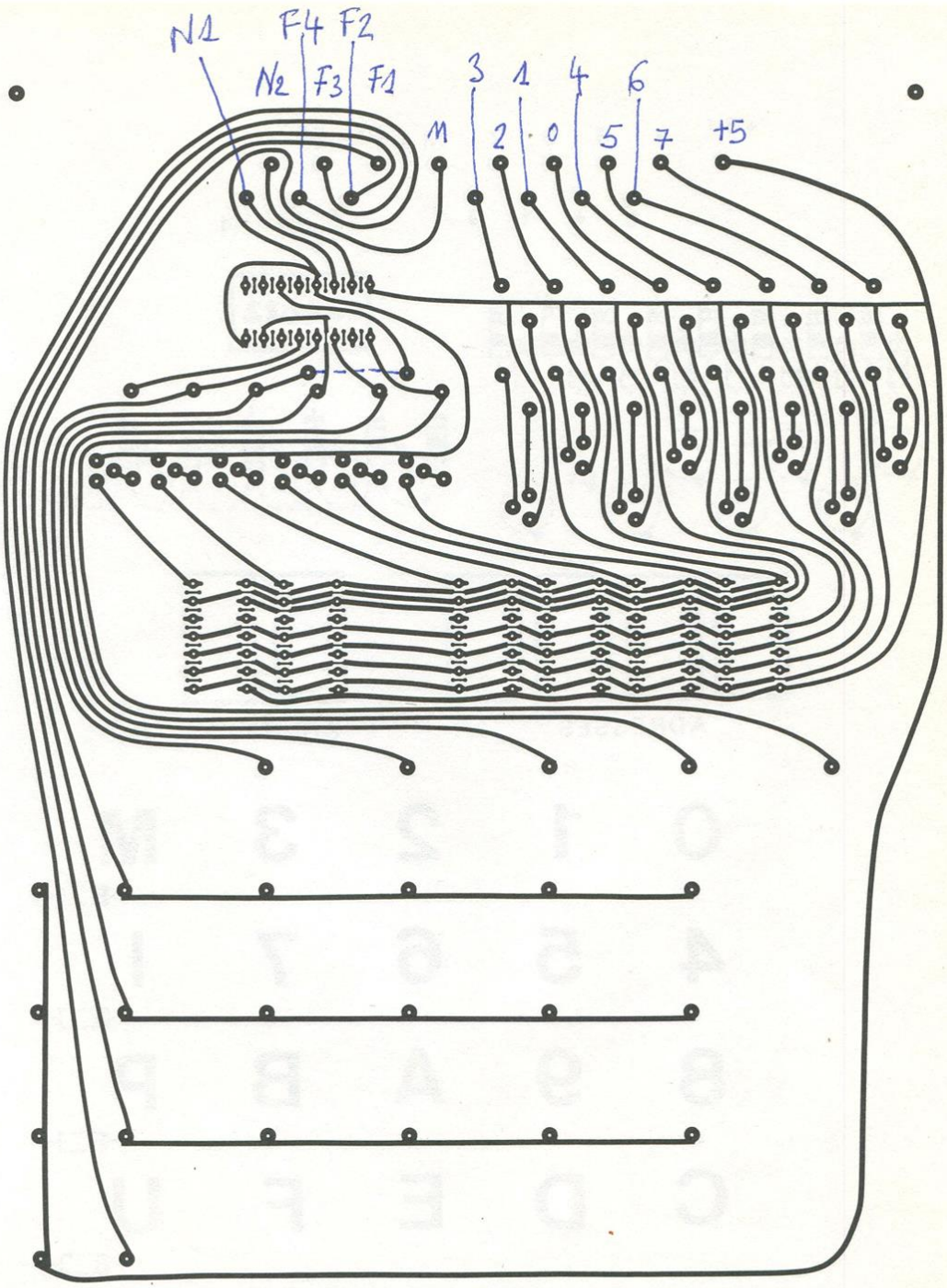
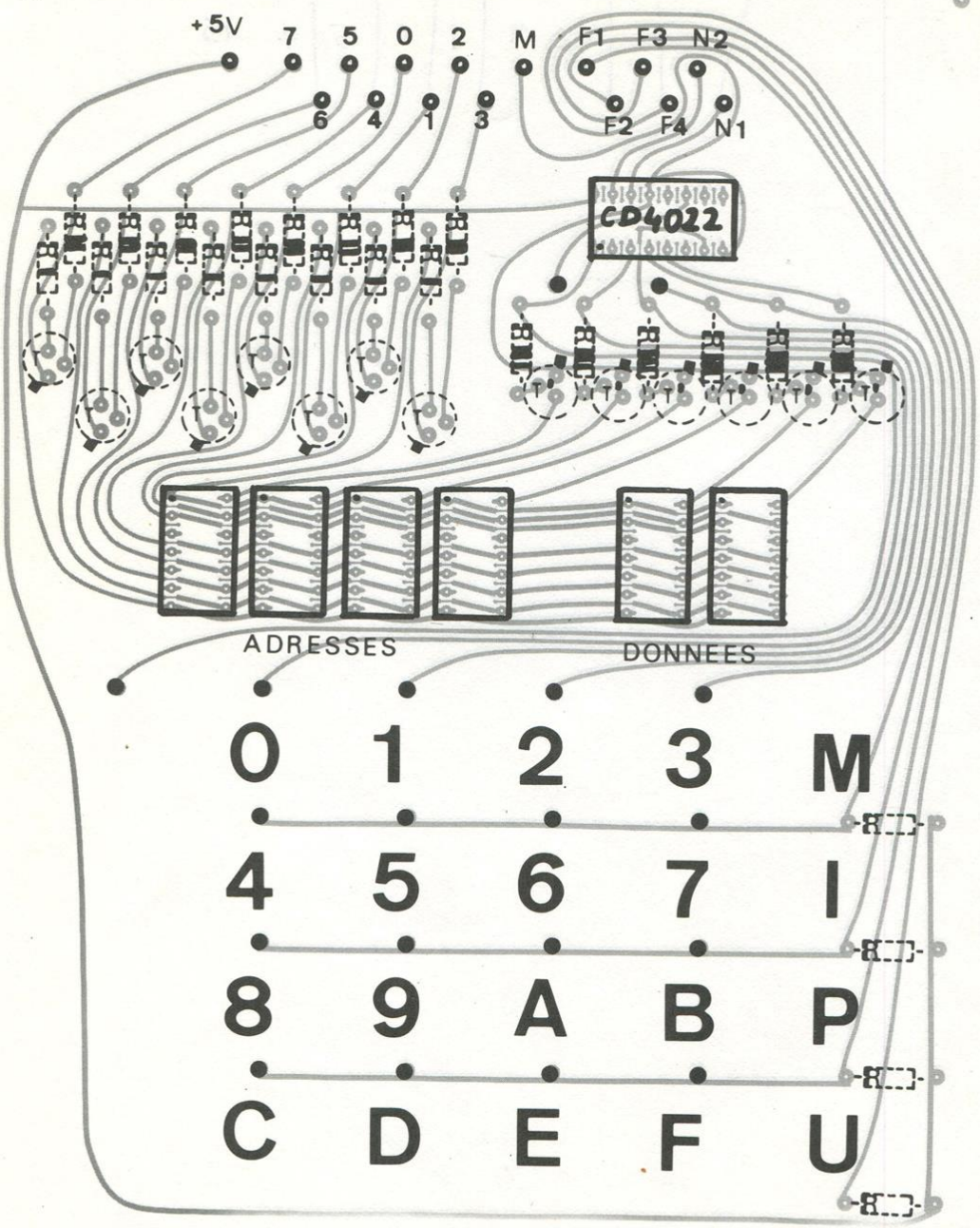


Fig. III. 13. Circuit imprimé face dessous



VILÉMIO 3 *Fig. III. 14. Implantation des composants*

— on soude ensuite les autres supports sans être obligé de respecter un ordre particulier ;

— on soude ensuite les composants passifs, les diodes, les transistors et la LED ;

— on place les interrupteurs et on les câble comme l'indique la figure III. 10 ;

— il ne reste plus qu'à souder les picots à souder. On orientera ces picots de telle sorte qu'ils ne puissent court-circuiter les pistes du circuit imprimé de la face dessus.

Nous rappelons que le montage demande du soin. Il faut utiliser un fer à souder à pointe fine, ne pas oublier de soudures, ne pas court-circuiter des pistes.

Bon courage !

II. 3. Matériel nécessaire à la réalisation de la maquette « clavier-afficheur »

— un compteur de Johnson CD 4022 (RCA) ;
— un support à souder de 16 broches ;
— 6 afficheurs HP 5082-7653 (Hewlett Packard) ;
— 8 transistors 2 N 2904 ;
— 6 transistors 2 N 2222 ;
— 4 résistances de 820 kΩ ;
— 8 résistances de 180 Ω ;
— 8 résistances de 4,7 kΩ ;
— 6 résistances de 2,7 kΩ ;
— des picots à souder.

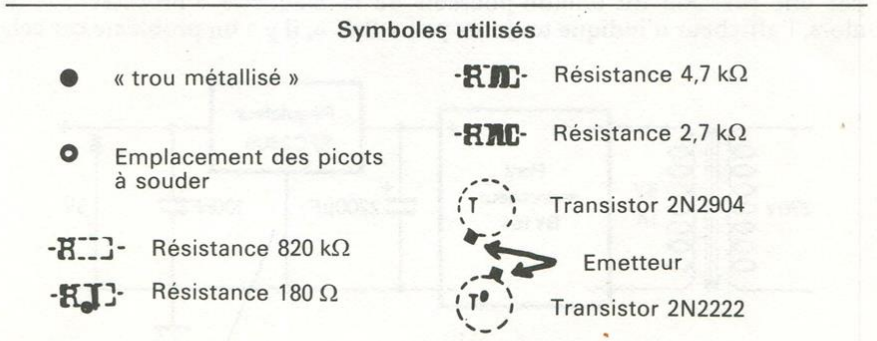


Fig. III. 15. Symboles utilisés sur la figure III-14

II. 4. Réalisation pratique de la maquette « clavier-afficheur »

Le dessin du circuit imprimé face dessus, c'est-à-dire le dessin du circuit tel qu'on le voit la maquette posée à l'endroit sur une table, est donné à la figure III. 12.

Le dessin du circuit imprimé face dessous, c'est-à-dire tel qu'on le voit la maquette étant placée retournée sur une table, est donné figure III. 13.

Le schéma d'implantation des composants est donné à la figure III. 14, et l'explication des symboles utilisés figure III. 15.

La réalisation pratique de la maquette « clavier-afficheur » ne présente pas de difficulté particulière. On fera attention à ce que les boîtiers des transistors T 2 N 2904 ne se touchent pas car ceux-ci sont reliés aux collecteurs.

Lorsque vous aurez terminé la réalisation des maquettes « mise au point » et « clavier-afficheur », et examiné soigneusement l'état (qui doit être bon) des soudures, il faudra les connecter entre elles comme le montre la figure III. 7. Les connexions peuvent être faites avec du fil soudé car normalement ces deux maquettes forment un tout indivisible.

II. 5. Les premiers essais

Les circuits intégrés étant placés sur leur support et les maquettes « processeur », « mise au point » et « clavier-afficheur » reliées entre elles, il est temps de vérifier le bon fonctionnement de l'ensemble. L'ensemble doit être alimenté, par l'intermédiaire des deux douilles de la maquette « processeur », par une source de tension + 5 V pouvant fournir un courant de 250 mA ; cependant, il vaut mieux prévoir une source de tension + 5 V capable de délivrer 1 A. Son schéma est donné à la figure III. 16 à titre indicatif.

Immédiatement après la mise sous tension de votre système, l'afficheur doit indiquer « Prêt ». Si cela n'est pas le cas, effectuer une remise à zéro par une pression du bouton-poussoir de la maquette « processeur ». Si alors, l'afficheur n'indique toujours pas « Prêt », il y a un problème car cela

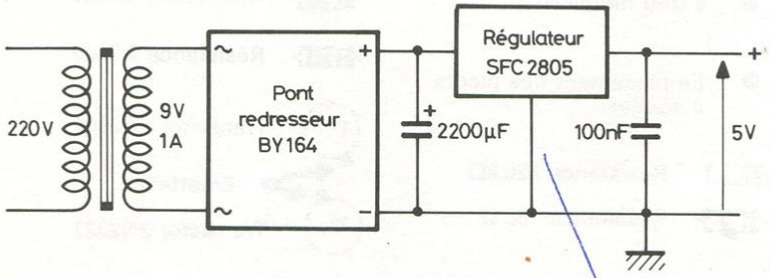


Fig. III. 16. Exemple d'alimentation

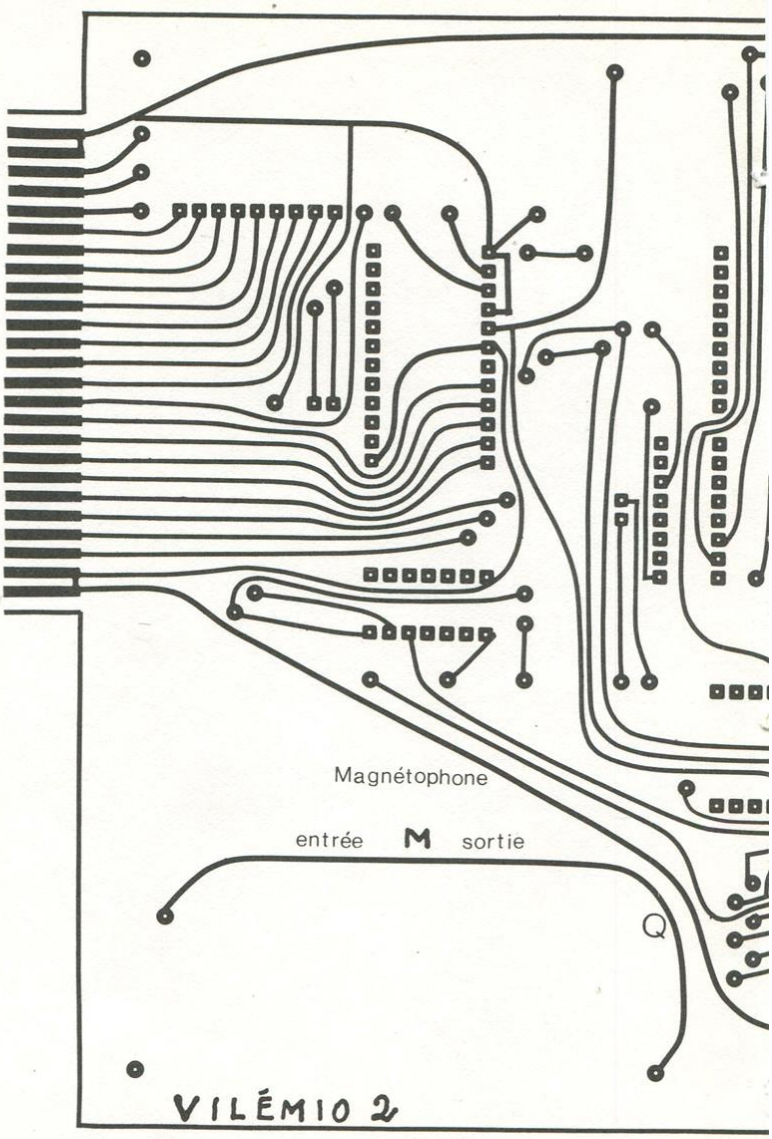
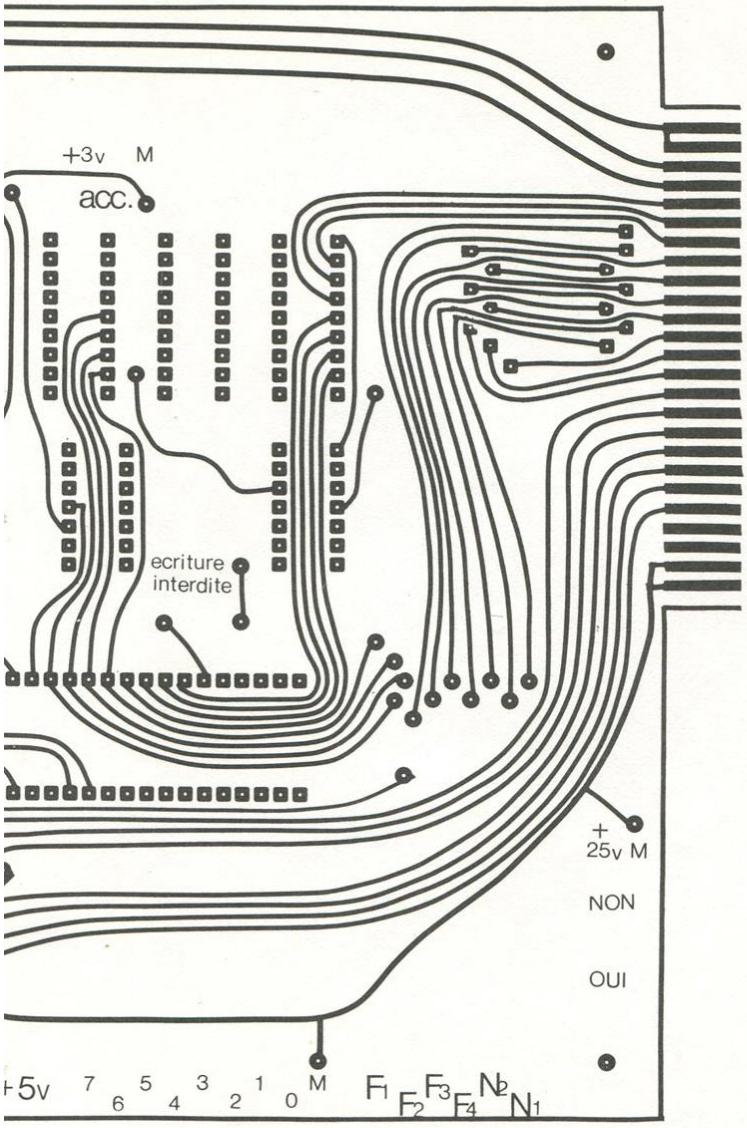


Fig. III. 8. Circuit imprimé de la maquette « mise au point » face dessus (composants)



(Voir pages 144-145 côté cuivre).

ne marche pas ! Pour résoudre ce problème il n'y a qu'un moyen : examiner les soudures, faire celles qui ont été oubliées, refaire celles qui présentent un aspect douteux, éliminer les éventuels courts-circuits entre les pistes du circuit imprimé, éliminer les bavures de métal qui pourraient encore subsister si le perçage a été effectué avec un foret mal affuté... La seule source de panne est donc une exécution peu soignée du montage des maquettes ; nous supposons naturellement que les circuits intégrés n'ont pas été montés à l'envers et que l'EPROM moniteur est effectivement programmée...

Remarque importante

Avant de mettre le système sous tension, et si un accumulateur n'est pas placé entre les bornes marquées + 3 V et M, il *faut* relier ces deux bornes par un cordon.

Chapitre 4

CONCEPTION ET RÉALISATION PRATIQUE DE LA MAQUETTE « ENTRÉE/SORTIE »

I. Les instructions OUTPUT et INPUT du microprocesseur CDP 1802

Comme leur nom l'indique les instructions OUTPUT et INPUT permettent de sortir ou d'entrer des données vers, ou venant, du monde extérieur. Lorsqu'on examine les opérations effectuées lors du cycle d'exécution de ces instructions on s'aperçoit que les sorties N_2, N_1, N_0 du microprocesseur (appelées lignes N) sont activées. La combinaison binaire qui apparaît sur les lignes N est égale au numéro de l'instruction OUTPUT ou INPUT utilisée ; par exemple, l'instruction OUT 3 fournit la combinaison (N_2, N_1, N_0) = (0, 1, 1), et l'instruction INP 6 fournit la combinaison (N_2, N_1, N_0) = (1, 1, 0).

L'existence des lignes N invite le concepteur à les utiliser pour valider des boîtiers destinés au couplage entre le système à microprocesseur et le monde extérieur. Comme le montre la figure IV. 1, nous avons choisi d'utiliser pour ces boîtiers des registres CDP 1852 dont le brochage est donné figure IV. 2.

II. Couplage du système avec le monde extérieur

Un registre CDP 1852 est constitué par 8 bascules « latch » transparentes lorsque l'entrée Cl. du boîtier est au niveau logique 1. Lorsque l'entrée Cl. revient au niveau logique 0, le registre est « verrouillé », c'est-à-dire qu'il conserve en mémoire la donnée qu'on vient de lui fournir lorsque Cl. était au niveau 1.

Un registre CDP 1852 peut fonctionner selon deux modes distincts :

II. 1. Le mode « sortie »

Dans ce mode, l'entrée Cl. du registre est active si et seulement si les entrées $\overline{CS_1}$ et CS_2 sont respectivement au niveau logique 0 et 1. Examinez la figure IV. 1 : l'entrée CS_2 du registre de sortie est reliée à la sortie OUT 1 du décodeur CDP 1853 dont le brochage est donné figure IV. 3. Le ta-

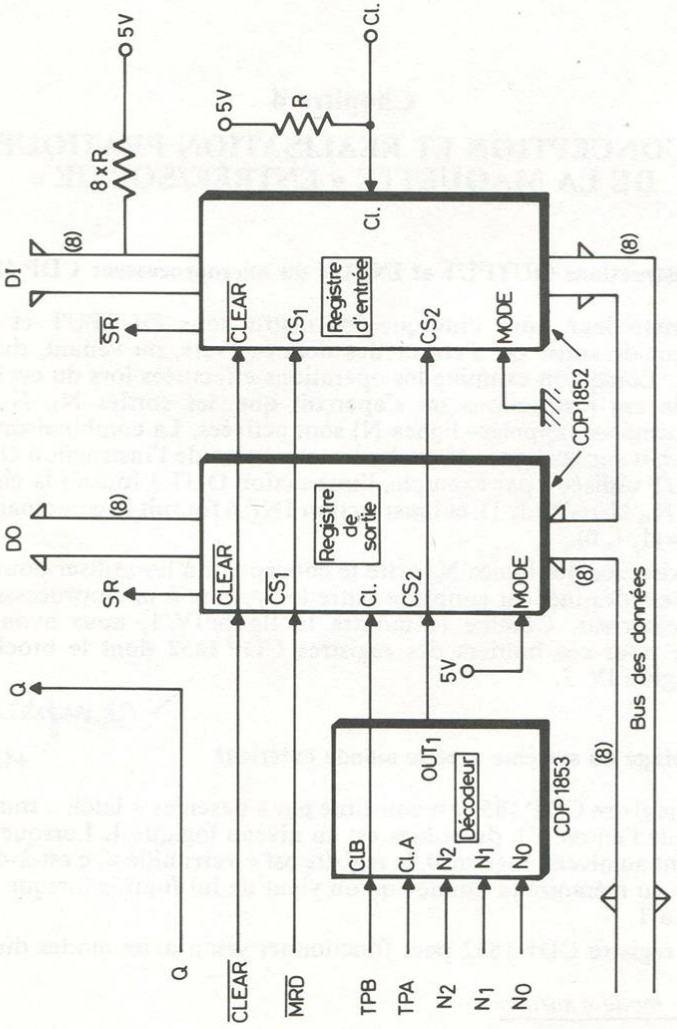


Fig. IV. 1. Schéma synoptique de la maquette « entrée-sortie »

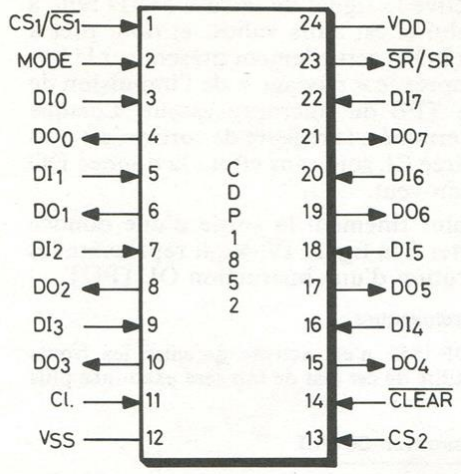


Fig. IV. 2. Brochage du CDP 1852

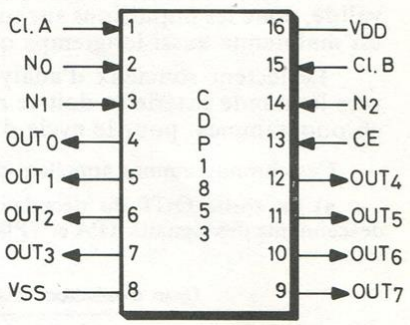


Fig. IV. 3. Brochage du CDP 1853

bleau IV. 1 indique la fonction de ce décodeur destiné d'une façon générale à valider un boîtier de sortie parmi sept, et ceci selon la combinaison présente sur les lignes N_2, N_1, et N_0. La sortie OUT1 de ce décodeur est au niveau logique 1 lorsque (N_2, N_1, N_0) = (0, 0, 1) ; par conséquent, lors de l'exécution de l'instruction OUT1 (de code 61), l'entrée CS_2 du registre de sortie est au niveau logique 1. En outre, lors de l'exécution de l'instruction OUT1, le mot présent dans la mémoire à l'adresse définie par le registre R (X) interne au microprocesseur, est placé sur le bus des données ; cette

			Sorties							
N_2	N_1	N_0	0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Tableau IV.1.

opération de *lecture* de la mémoire active le signal de lecture $\overline{MRD}$ relié à l'entrée $\overline{CS_1}$ du registre de sortie : celui-ci est alors validé, et dont prêt à transférer vers sa sortie DO le mot M (R(X)) actuellement présent sur le bus des données ; ce transfert est effectif après le « passage » de l'impulsion de déclenchement fournie par la sortie TPB du microprocesseur. Lorsque l'exécution de l'instruction OUT1 est terminée, le registre de sortie n'est plus validé, donc les impulsions sur son entrée Cl. sont sans effet : la donnée DO est maintenue aussi longtemps que l'on veut.

Le lecteur soucieux d'analyser plus finement la sortie d'une donnée vers le monde extérieur, doit se reporter à la figure IV. 4 qui représente les chronogrammes pour le cycle d'exécution d'une instruction OUTPUT.

Ces chronogrammes appellent deux remarques :

a) La sortie OUT du décodeur CDP 1853 n'est activée qu'entre les fronts descendants des signaux TPA et TPB ; l'utilité de cet état de fait sera examinée plus loin.

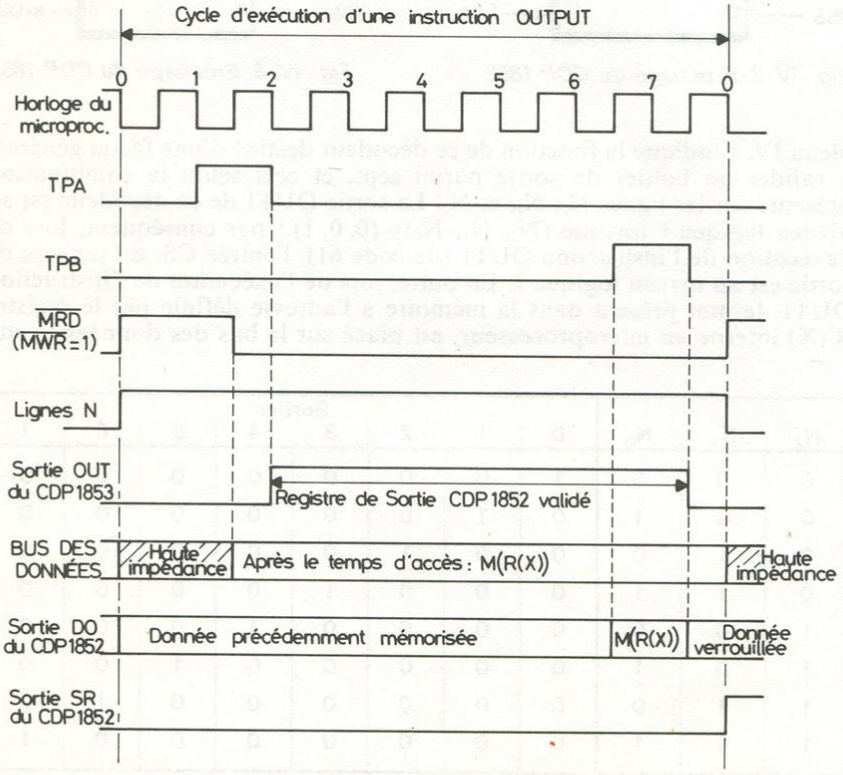


Fig IV. 4. Chronogrammes d'une instruction OUTPUT

b) le registre CDP 1852 présente une sortie S.R. (S.R. comme Service Request en anglais, ou requête de service en français). S.R. passe au niveau logique 1 lorsque la donnée a effectivement été transférée en sortie DO ; le signal S.R. peut donc servir à signaler à un système électronique qu'une donnée DO est prête à être utilisée en vue de l'exécution d'un certain service.

II. 2. Le mode « entrée »

Dans ce mode, la sortie effective du registre CDP 1852 (sortie notée DO_{int}, c'est-à-dire DO interne ; voyez la figure IV. 5) est reliée au bus des

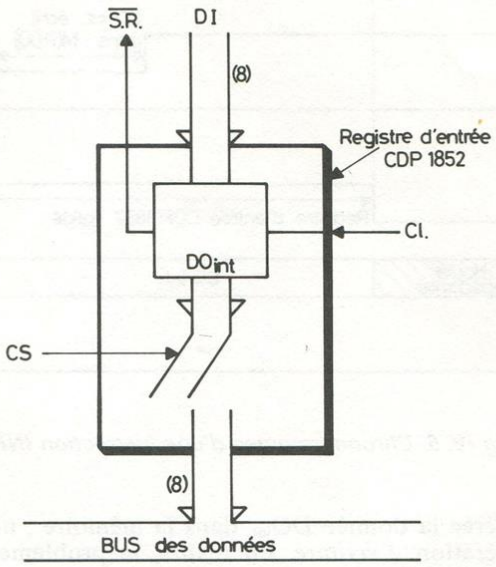


Fig. IV. 5. Le registre 3 états CDP 1852

données par l'intermédiaire d'une logique 3 états symbolisée par un interrupteur. Si l'interrupteur est fermé, DO_{int} est placé sur le bus des données ; si l'interrupteur est ouvert, le boîtier est en haute impédance. L'interrupteur électronique est commandé par $CS = CS_1 . CS_2$: le boîtier est validé (interrupteur fermé) si $CS_1 = 1$ *et* $CS_2 = 1$. Comme le montre la figure 1, l'entrée CS_1 est reliée au signal de lecture $\overline{MRD}$ et l'entrée CS_2 à la sortie OUT1 du décodeur ; ainsi, lors de l'exécution de l'instruction INPUT1 de code 69, le mot noté DO_{int} contenu dans le registre d'entrée est placé sur le bus des données, ce qui permet son transfert dans la mémoire à l'adresse définie par R (X), et dans l'accumulateur D.

La figure IV. 6, qui présente les chronogrammes relatifs au cycle d'exécution d'une instruction INPUT, montre avec précision de quelle

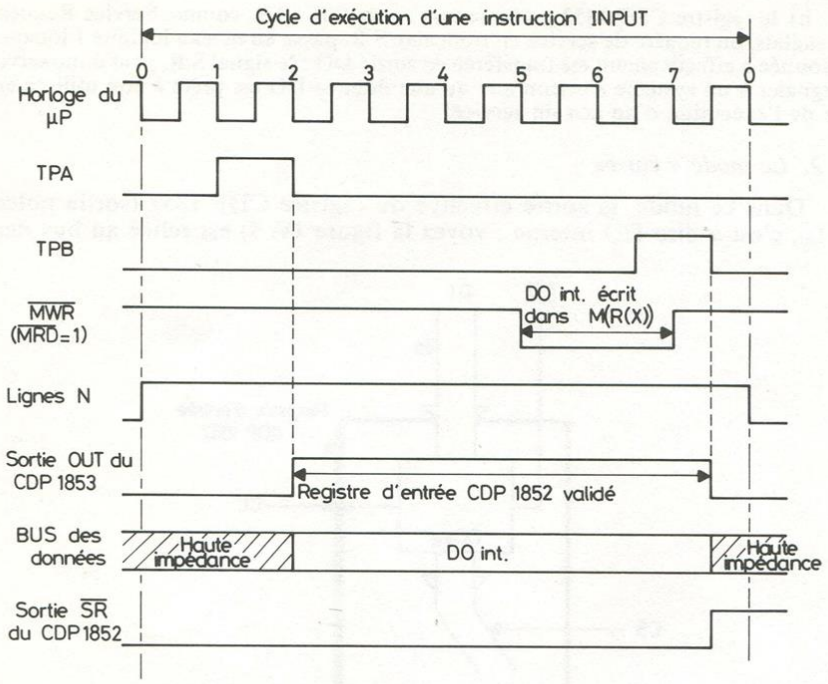


Fig IV. 6. Chronogrammes d'une instruction INPUT

façon est transférée la donnée $DO_{int.}$ dans la mémoire ; nous notons qu'il s'agit d'une opération d'*écriture.* En réalité, le problème qui est posé à l'utilisateur est de transférer vers la mémoire et le microprocesseur, non pas vraiment la donnée $DO_{int.}$ mais la donnée DI présente à l'entrée du registre CDP 1852 ; il a deux façons de traiter ce problème :

a) l'entrée Cl. du CDP 1852 est constamment au niveau logique 1. Comme nous avons câblé entre cette entrée Cl. et le +5V une résistance R de 33 kΩ, Cl. = 1 si on ne s'en occupe pas. Nous rappelons que dans ce cas le registre est transparent, c'est-à-dire que $DO_{int.} = DI$. Ainsi, lors de l'exécution de l'instruction INP1, c'est effectivement DI qui est transféré dans la mémoire et dans l'accumulateur D. Nous remarquons en passant, que la fonction « registre » n'est pas utilisée : notre boîtier CDP 1852 est réduit à un interrupteur ouvert ou fermé.

b) l'entrée Cl. est commandée par un système électronique externe, ou par un opérateur humain de façon manuelle (on peut alors imaginer un montage constitué par un bouton-poussoir associé à une logique anti-re-

bond). Examinons la figure IV. 7 qui représente la réponse du boîtier à une impulsion appliquée sur son entrée Cl.

— dans un état initial, Cl. est au niveau 0, et la sortie $\overline{S.R.}$ est au niveau 1. La donnée $DO_{int.}$ contenue dans le registre est celle précédemment mémorisée. Dans cet état initial, on supposera (ce qui est normalement le cas) que le boîtier n'est pas validé, c'est-à-dire que nous ne sommes pas dans le cycle d'exécution de l'instruction INP1.

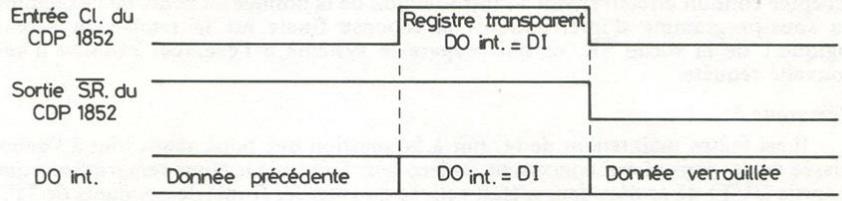


Fig. IV. 7. Réaction du CDP 1852 aux implusions d'horloge

— un front montant est envoyé sur l'entrée Cl., rendant le registre transparent ; ainsi, et tant que Cl. = 1, $DO_{int.} = DI$.

— cependant, la donnée DI n'est *mémorisée* dans le registre qu'après le front descendant de Cl. ainsi, l'entrée Cl. étant redescendue au niveau logique 0, la donnée $DO_{int.}$ est stable : le registre est verrouillé. Dès que la donnée DI est mémorisée dans le registre, sa sortie $\overline{S.R.}$ passe au niveau 0. Le signal $\overline{S.R.}$ peut donc servir à signaler au microprocesseur qu'une donnée est prête à être introduite dans le système. Généralement, on relie la sortie $\overline{S.R.}$ à l'entrée $\overline{INT}$ (interruption) du microprocesseur ; dans ce cas, la *réponse* à l'impulsion Cl. *provoquée par le monde extérieur,* est l'exécution d'un sous-programme d'interruption chargé d'entrer la donnée $DO_{int.}$ Lorsque la donnée est entrée, la figure 6 montre que la sortie SR revient au niveau logique 1, ce qui prépare le système à la réception d'une donnée suivante.

Remarque 1

On dispose donc au moins de deux façons d'entrer une donnée vers le système à microprocesseur :

— dans la procédure pour laquelle Cl. est constamment au niveau 1, il faut remarquer que rien ne signale au microprocesseur qu'une donnée DI stable est présente en entrée du boîtier CDP 1852. Ainsi on ne peut généralement pas bien prévoir à quel moment cette donnée sera transférée vers le système. On dit alors que cette donnée est transférée dans un mode *asynchrone.* Cette méthode a cependant l'avantage d'être simple à mettre en œuvre. Elle nécessite pourtant, si la donnée DI est susceptible de varier, un examen *périodique* (effectué par un programme de structure bouclée) de l'état logique de DI.

— dans la procédure pour laquelle CI. est commandée par le monde extérieur, et génère un signal d'interruption, on peut dire que la donnée est transférée vers le système dans un mode *synchrone*, puisque ce transfert est opéré à une époque parfaitement déterminée par rapport au front descendant de CI. Dans la littérature américaine, cette méthode est appelée technique du « handshaking », ce qui veut dire « échange de poignées de main » ; ce terme pour le moins imagé exprime prosaïquement le concept « requête-réponse à la requête » rencontré couramment dans les systèmes à microprocesseurs : le front descendant de CI. établit en fait une requête d'interruption en vue d'introduire une donnée dans le système ; la requête acceptée conduit effectivement à l'introduction de la donnée au cours de l'exécution du sous-programme d'interruption ; la réponse finale est la remise au niveau logique 1 de la sortie $\overline{SR}$, ce qui prépare le système à l'éventuel examen d'une nouvelle requête.

Remarque 2

Il est temps maintenant de revenir à la question que nous avons tout à l'heure laissée en suspens et qui concernait le décodeur CDP 1853. Nous remarquions que la sortie OUT1 de ce décodeur n'était activée qu'entre les fronts descendants de TPA et TPB. Pourquoi ?

Observez les figures 4 et 6 et supposez que la sortie OUT1 soit activée dès le début des cycles d'exécution des instructions OUTPUT et INPUT. Imaginez qu'on demande alors au microprocesseur d'exécuter l'instruction OUT1 de code 61 : vous vous apercevez alors qu'au début du cycle, ce n'est pas le registre de sortie qui est validé mais le registre d'entrée puisqu'à ce moment $\overline{MRD} = 1$; pour éviter cette sélection transitoire du registre d'entrée, il suffit de valider le registre de sortie seulement quand $\overline{MRD} = 0$ c'est-à-dire en particulier lors du front descendant de TPA.

Ce problème n'aurait pas existé si nous avions décidé de relier directement la sortie N_0 du microprocesseur à l'entrée CS_2 du registre de sortie, et la sortie N_1 par exemple à l'entrée CS_2 du registre d'entrée. Nous avons évité de procéder de cette façon car alors l'entrée d'une donnée aurait perturbé l'affichage au niveau de la maquette « clavier-afficheur » ; rappelons en effet que celle-ci utilise les lignes N_1 et N_2.

III. Réalisation pratique de la maquette « entrée/sortie », dite VILÉMIO 4

III. 1. Schéma de câblage

Celui-ci est donné figure IV. 8. La seule remarque à faire est que les entrées DI_7 à DI_0 sont reliées au + 5 V par l'intermédiaire de résistances. Ainsi, si l'on ne met « rien » sur ces entrées, elles sont toutes au niveau logique 1.

III. 2. Matériel nécessaire

— 2 registres CDP 1852 CE (RCA),
— 1 décodeur CDP 1853 CE (RCA),
— 2 supports de 24 broches à wrapper,
— 1 support de 16 broches à wrapper,

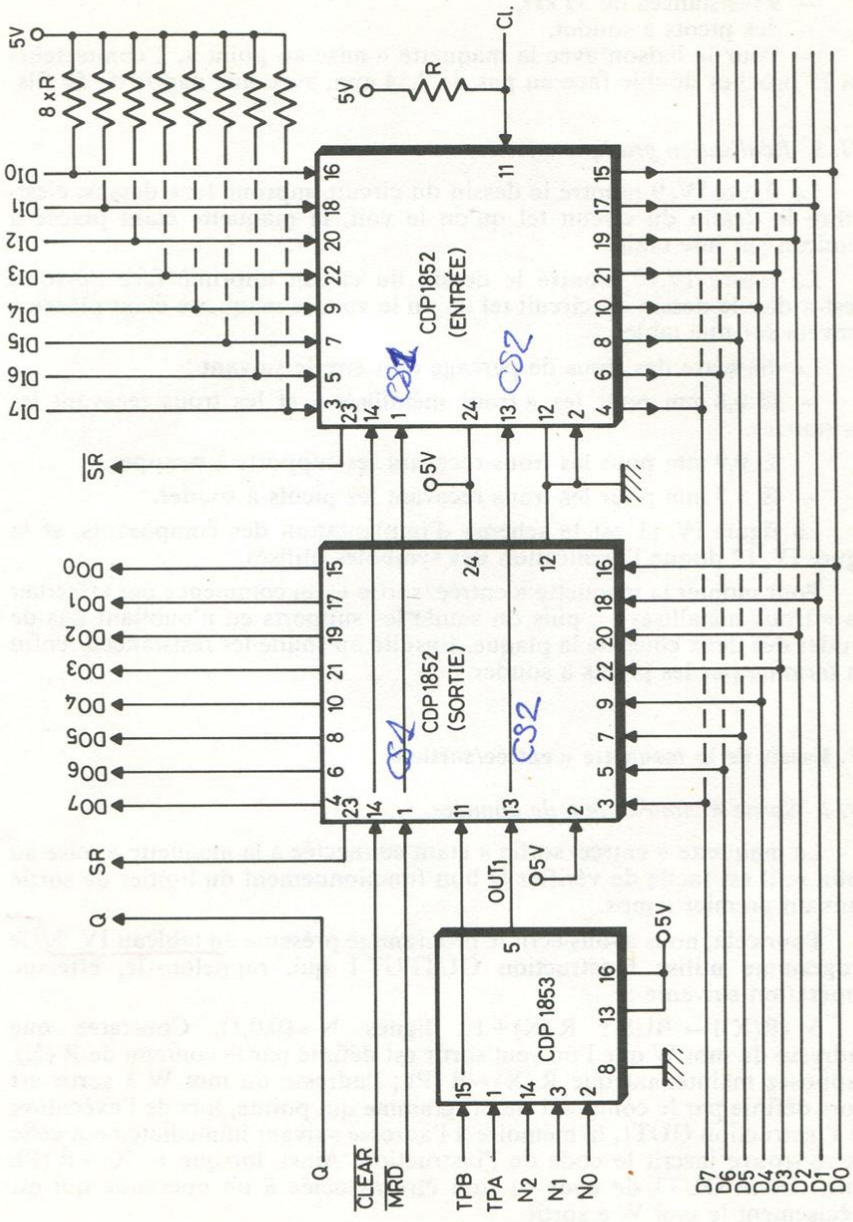


Fig. IV. 8. Schéma de câblage de la maquette « entrée-sortie »

— 9 résistances de 33 kΩ,
— des picots à souder,
— Pour la liaison avec la maquette « mise au point », 2 connecteurs 2 × 25 broches double face au pas de 2,54 mm, avec une nappe de 50 fils.

III. 3. Réalisation pratique effective

La figure IV. 9 montre le dessin du circuit imprimé face dessus, c'est-à-dire le dessin du circuit tel qu'on le voit, la maquette étant placée à l'endroit sur une table.

La figure IV.10 montre le dessin du circuit imprimé face dessous, c'est-à-dire le dessin du circuit tel qu'on le voit, la maquette étant placée à l'envers sur une table.

Le diamètre des trous de perçage doit être le suivant :

— ∅ 0,8 mm pour les « trous métallisés » et les trous recevant les résistances.

— ∅ 0,9 mm pour les trous recevant les supports à wrapper.

— ∅ 1,3 mm pour les trous recevant les picots à souder.

La figure IV. 11 est le schéma d'implantation des composants, et la figure IV. 12 donne l'explication des symboles utilisés.

Pour monter la maquette « entrée/sortie », on commence par effectuer les « trous métallisés » ; puis on soude les supports en n'oubliant pas de souder des deux côtés de la plaque. Ensuite on soude les résistances, enfin on termine par les picots à souder.

IV. Essais de la maquette « entrée/sortie »

IV. 1. Sortie « immédiate » de données

La maquette « entrée/sortie » étant connectée à la maquette « mise au point », il est facile de vérifier le bon fonctionnement du boîtier de sortie dans un premier temps.

Pour cela, nous avons écrit le programme présenté au tableau IV. 2. Ce programme utilise l'instruction OUTPUT 1 qui, rappelons-le, effectue l'opération suivante :

M (R(X)) → BUS ; R (X) + 1 ; lignes N = (0,0,1). Constatez que l'adresse du mot W que l'on veut sortir est définie par le contenu de R (X). Supposez maintenant que R (X) = R (P) ; l'adresse du mot W à sortir est alors définie par le compteur de programme qui pointe, lors de l'exécution de l'instruction OUT1, la mémoire à l'adresse suivant immédiatement celle où se trouve inscrit le code de l'instruction. Ainsi, lorsque R (X) = R (P), l'instruction OUT1 de code 61 doit être associée à un opérande qui est précisément le mot W à sortir.

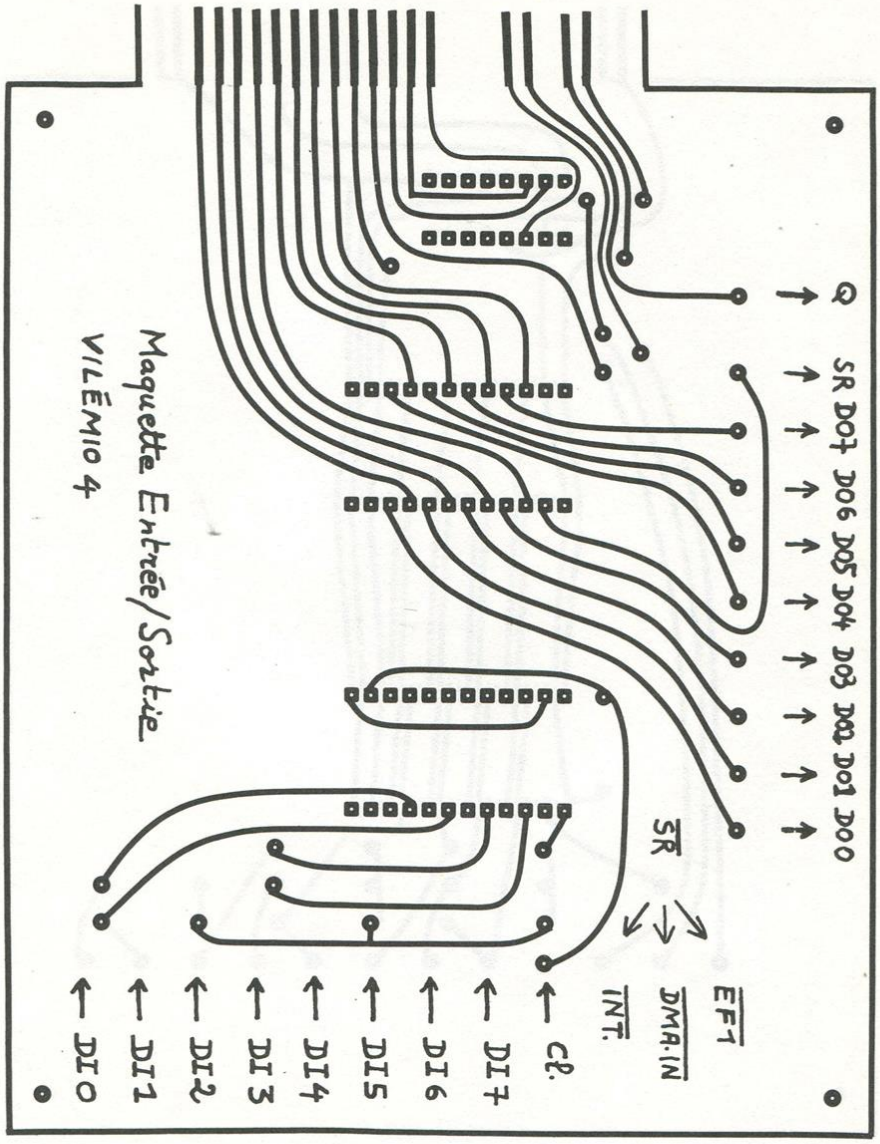


Fig. IV. 9. Circuit imprimé de la maquette « entrée-sortie » face dessus

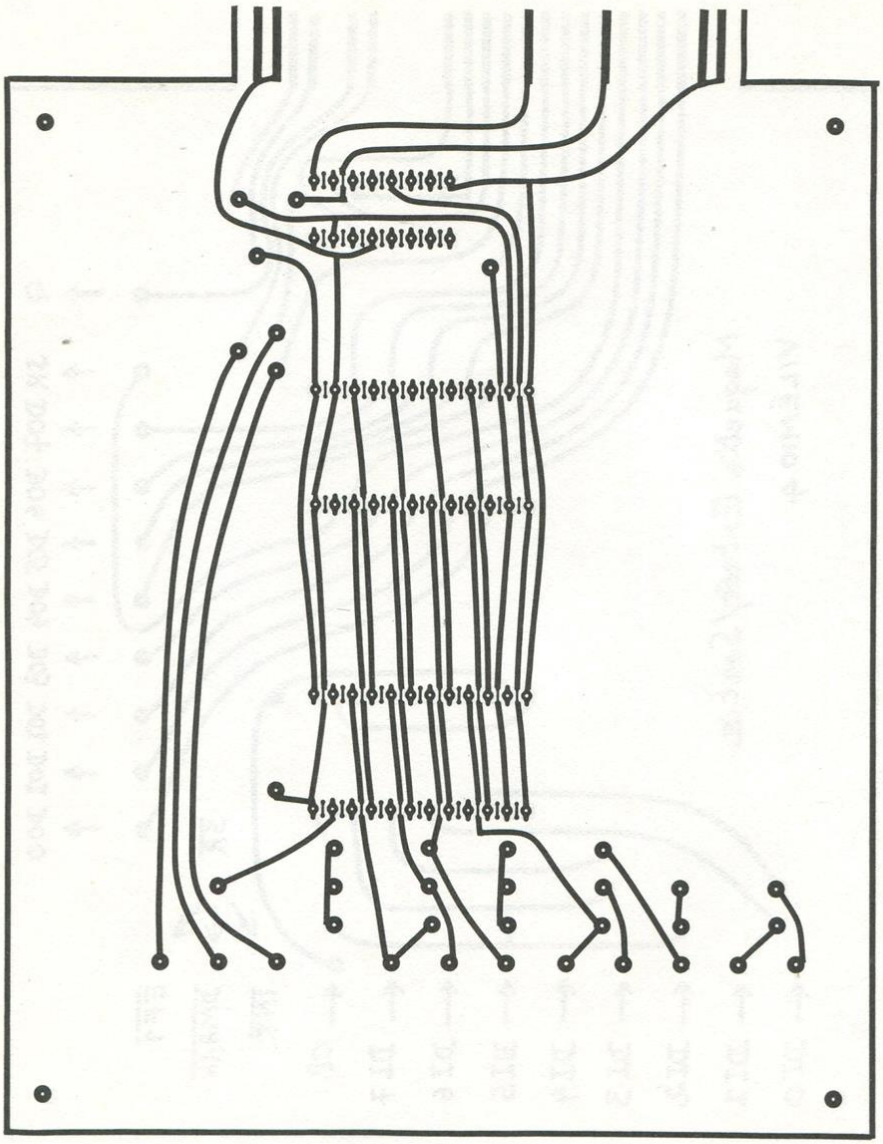


Fig. IV. 10. Circuit imprimé face dessous

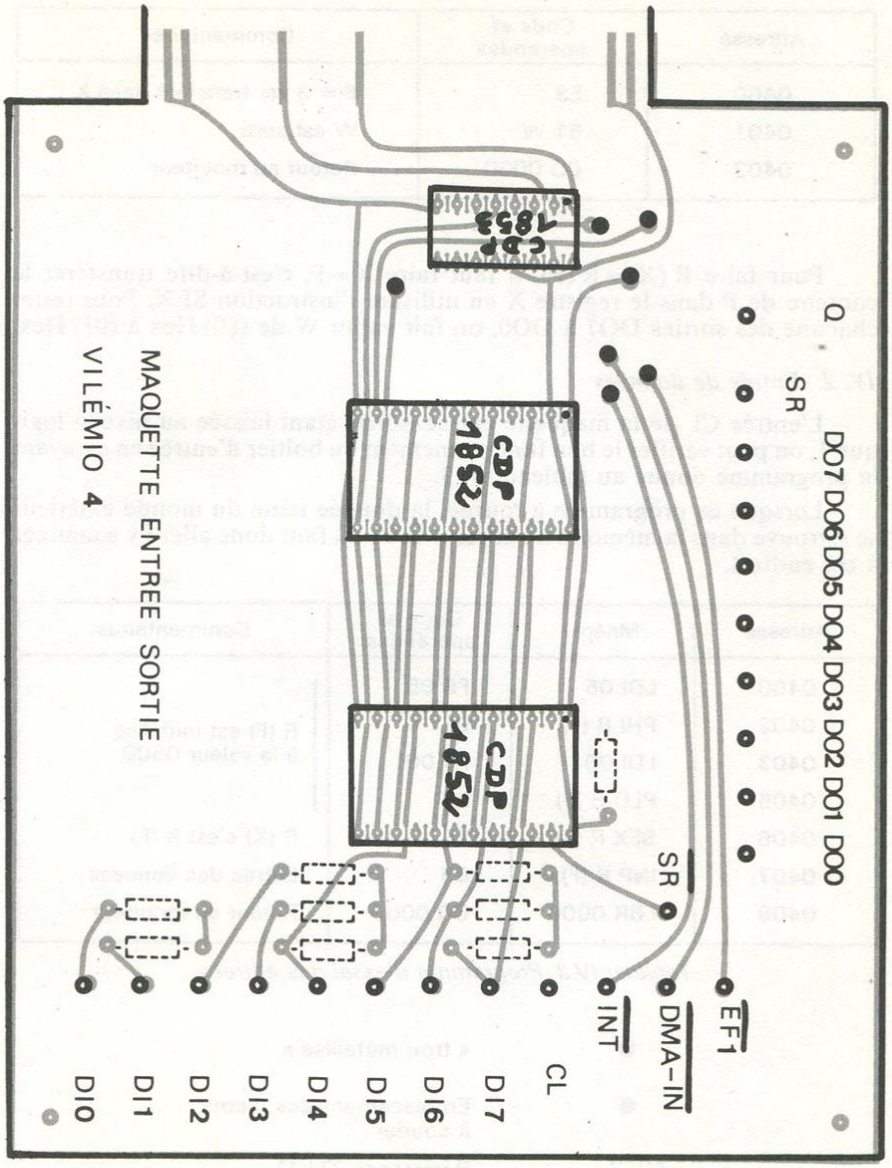


Fig. IV. 11. Plan d'implantation des composants

IV. 2

Adresse	Code et opérandes	Commentaires
0400	E3	P = 3 est transféré dans X
0401	61 W	W est sorti
0403	CO 0000	Retour au moniteur

Pour faire R (X) = R (P), il faut faire X = P, c'est-à-dire transférer le contenu de P dans le registre X en utilisant l'instruction SEX. Pour tester chacune des sorties DO7 à DO0, on fait varier W de (80) Hex à (01) Hex.

IV. 2. Entrée de données

L'entrée Cl. de la maquette entrée/sortie étant laissée au niveau logique 1, on peut vérifier le bon fonctionnement du boîtier d'entrée en essayant le programme donné au tableau IV. 3.

Lorsque ce programme a tourné, la donnée issue du monde extérieur se retrouve dans la mémoire à l'adresse 0500 ; il faut donc aller l'y examiner à cet endroit.

Adresse	Mném.	Code et opérandes	Commentaires
0400	LDI 05	F8 05	R (F) est initialisé à la valeur 0500
0402	PHI R (F)	BF	
0403	LDI 00	F8 00	
0405	PLO R (F)	AF	
0406	SEX R (F)	EF	R (X) c'est R (F)
0407	INP R (F)	69	Entrée des données
0408	LBR 0000	CO 0000	Retour au moniteur

Tableau IV.3. Programme d'essai des entrées

Fig. IV. 12. Symboles utilisés sur la figure IV-11

Deuxième partie

Mode d'utilisation et conception logicielle du système VILÉMIO Ses compléments : interface cassette et programmateur d'EPROM

Chapitre 5

FONCTION ET PROCÉDURE D'UTILISATION DU MONITEUR

L'exécution d'un programme par le microprocesseur exige que ce programme soit préalablement chargé dans la mémoire associée au microprocesseur. Les deux modules complémentaires intervenant dans la structure matérielle de base (module mise au point et interfaces, module clavier et afficheur) ont été conçus pour faciliter l'enregistrement, la mise au point, la vérification et le lancement d'un programme de l'utilisateur. La gestion de toutes les opérations nécessaires est confiée au microprocesseur du module processeur lui-même, en association avec une EPROM contenant un programme spécifique appelé *programme moniteur.*

Le présent chapitre a pour but d'indiquer au lecteur quelle aide il peut attendre du moniteur et lui en préciser le mode d'emploi. Nous reportons aux chapitres suivants la description et l'explication matérielles et logicielles du moniteur pour ne nous consacrer ici qu'à sa *fonction.*

I. Description générale — Mise en route

Le clavier à vingt touches sensitives et l'afficheur à six digits (sept segments et un point) constituent de véritables périphériques respectivement d'entrée et de sortie qui permettent la communication de l'utilisateur avec le système (figure V. 1).

En plus des seize touches de caractères hexadécimaux (0 à F), le clavier comporte quatre touches permettant la commande des fonctions suivantes :

M : examen et modification éventuelle du contenu d'une ligne de la *M*émoire

I : Initialisation du moniteur

P : exécution d'un *P*rogramme *ou* examen de la ligne suivante (ou *P*as suivant)

U : fonction choisie et programmée par l'*U*tilisateur.

Les quatre digits de gauche de l'afficheur sont essentiellement destinés à indiquer une *adresse* de la mémoire, adresse où l'on désire examiner le contenu de la mémoire ou bien adresse du début d'un programme que l'on veut exécuter. Les deux digits de droite indiquent une *donnée.*

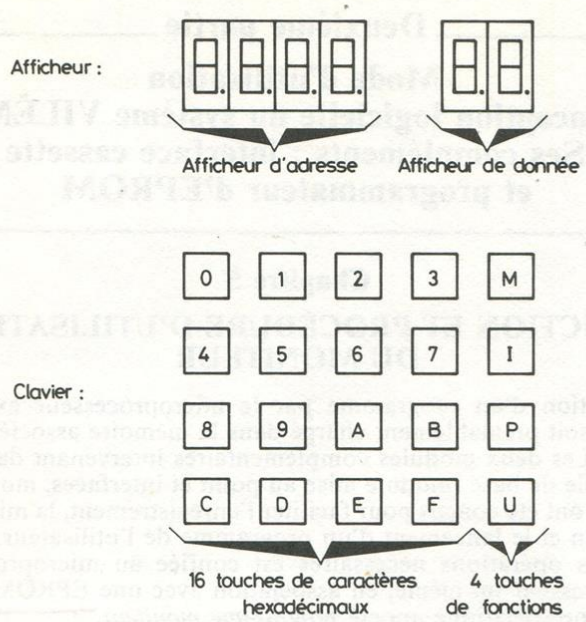


Fig. V. 1. Afficheur et clavier

Les connexions entre les trois modules de base étant effectuées et l'alimentation en continu réalisée, la mise en service du moniteur est assurée automatiquement, ou obtenue en agissant sur le poussoir de remise à zéro de la maquette processeur. Il s'effectue alors une procédure d'initialisation qui conduit à l'affichage du mot « PrEt » sur les quatre digits réservés normalement aux adresses. Les deux digits de données demeurent éteints. Le moniteur est alors « prêt » à enregistrer une adresse.

II. Description des fonctions du moniteur

II. 1. Examen et modification éventuelle du contenu d'une ligne de la mémoire

a) Pour examiner le contenu de la mémoire à l'adresse XYZT (code hexadécimal), l'utilisateur agit successivement sur les touches X, Y, Z et T du clavier. L'adresse XYZT se forme sur les digits d'affichage d'adresse, à partir de la droite, en « chassant » le mot initial PrEt vers la gauche. En cas d'erreur de frappe, il convient de recommencer jusqu'à ce que l'afficheur d'adresse indique bien XYZT. Le contenu M (XYZT) de la mémoire à l'adresse XYZT apparaît sur les deux digits de donnée après action et relâchement de la touche de fonction M.

b) Si la donnée lue ne convient pas, on peut alors la modifier. Si l'on désire écrire le mot $\alpha\,\beta$ (code hexadécimal) à l'adresse XYZT indiquée par l'afficheur, il suffit d'agir successivement sur les touches α et β du clavier. La donnée $\alpha\,\beta$ se forme sur les digits d'affichage de donnée, à partir de la droite, en « chassant » le mot M (XYZT) initial vers la gauche, *en même temps que cette donnée s'inscrit* dans la mémoire à l'adresse XYZT. En cas d'erreur de frappe, il convient de recommencer jusqu'à ce que l'afficheur de donnée indique bien $\alpha\,\beta$.

Attention : La modification de la donnée ne peut se faire que si la ligne mémoire adressée appartient à une RAM. Dans le cas d'une ROM, on peut lire la donnée écrite à l'adresse XYZT ; une tentative de modification de cette donnée amène bien une modification de l'afficheur de donnée mais n'altère pas le contenu de la mémoire : M (XYZT) ne peut pas être changé.

II. 2. Initialisation du moniteur

Quand on a terminé l'examen et la modification éventuelle d'une ligne de la mémoire (ou d'une suite de lignes adjacentes : voir plus loin), on peut quitter cette fonction en agissant sur la touche I. La procédure d'initialisation du moniteur reprend et conduit à l'affichage du mot PrEt (sous-entendu : à enregistrer une adresse).

II. 3. Examen et modification éventuelle du contenu d'une suite de lignes adjacentes de la mémoire

Il est possible d'utiliser la procédure décrite aux deux paragraphes précédents, pour chaque ligne (examen et modification du contenu d'une ligne, initialisation du moniteur, examen et modification du contenu de la ligne suivante dont on aurait aussi tapé l'adresse sur le clavier, initialisation du moniteur, etc.). Cette procédure est cependant très fastidieuse. Le programme moniteur autorise une procédure plus commode et rapide.

Quand l'afficheur indique PrEt, l'utilisateur agit sur les touches X, Y, Z et T du clavier pour afficher l'adresse de la première des lignes successives de la mémoire à examiner (comme au paragraphe 1). Il appuie ensuite sur la touche M pour examiner et éventuellement modifier le contenu M (XYZT) (comme au paragraphe 1). Ce travail terminé, il suffit de presser la touche P pour que l'afficheur indique l'adresse XYZT+1 de la ligne suivante ainsi que son contenu que l'on peut éventuellement modifier en utilisant les touches de caractères hexadécimaux. On passe à l'examen et à la modification éventuelle de la ligne d'adresse XYZT+2 en pressant une nouvelle fois la touche P, et ainsi de suite jusqu'à la dernière ligne à examiner.

Quand ces examens et modifications éventuelles sont terminés, on quitte cette fonction en agissant sur la touche I. Le moniteur se réinitialise et l'afficheur indique PrEt.

On comprend que cette procédure permet l'enregistrement rapide et la vérification aisée de tout un programme.

II. 4. Exécution d'un programme

Pour que le microprocesseur exécute un programme enregistré dans la mémoire à partir de l'adresse XYZT, l'utilisateur doit d'abord fournir cette adresse au système « prêt » à le recevoir (moniteur réinitialisé) en agissant sur les touches X, Y, Z et T du clavier. Après s'être assuré que XYZT a bien remplacé PrEt sur l'afficheur d'adresse (en cas d'erreur il suffit de reprendre l'action sur les touches X, Y, Z et T sans prendre la peine de réinitialiser par action sur la touche I), il suffit d'appuyer sur la touche P. L'afficheur (adresse et donnée) s'éteint et le microprocesseur exécute le programme à partir de l'adresse XYZT.

Attention : c'est au registre R (3) du microprocesseur que le programme moniteur confie le rôle de compteur du programme de l'utilisateur. Il convient donc d'éviter toute autre utilisation de ce registre dans le programme utilisateur (ou bien il faut préalablement changer de compteur de programme).

Remarque : quand le programme de l'utilisateur est en cours d'exécution, une action sur la touche I est impuissante à réinitialiser le moniteur car ce moniteur a perdu tout contrôle du système. On lui rend alors le contrôle du système et on le réinitialise en agissant sur le poussoir de remise à zéro de la maquette processeur.

II. 5. Fonction réservée à l'utilisateur

Pour donner plus de souplesse au système, il n'a pas été attribué de fonction particulière à la touche du clavier notée U (pour cela, il a fallu attribuer deux fonctions distinctes à la touche P : lancement d'un programme et incrémentation du pointeur de la mémoire en vue d'un examen ou d'une modification). Cependant, le programme moniteur est conçu pour qu'une pression de la touche U conduise à l'exécution d'une fonction prévue et programmée par l'utilisateur lui-même. Avant de demander l'exécution de cette fonction particulière, l'utilisateur doit écrire aux adresses 0C74 et 0C75 de la RAM de 128 octets, respectivement l'octet de poids fort et l'octet de poids faible de l'adresse du début du programme de cette fonction.

Cette faculté permet l'utilisation rapide d'un programme spécifique dont l'usage est courant pour l'utilisateur.

III. Illustration du fonctionnement et de l'emploi du moniteur

Afin de familiariser le lecteur — possédant ou non le matériel — à l'utilisation du moniteur nous proposons de suivre les différentes étapes conduisant à l'enregistrement, la vérification et l'exécution d'un programme élémentaire.

On se propose d'effectuer l'addition des contenus des lignes 0400 et 0401 de la mémoire et de stocker le résultat dans la ligne d'adresse 0402. Le

programme correspondant sera enregistré en mémoire RAM à partir de l'adresse 0500. Il apparaît sur la figure V. 2.

Adresse	Code	Opérande	Mnémonique	Commentaire
0500	E2		SEX R (2)	R (2) est pointeur de données
0501	F8		LDI	
0502		04		
0503	B2		PHI R (2)	R (2) est initialisé à 0400
0504	F8		LDI	
0505		00		
0506	A2		PLO R (2)	
0507	72		LDXA	M (0400) → D ; R (2) contient 0401
0508	F4		ADD	D + M (0401) → D : D contient le résultat
0509	12		INC R (2)	R (2) contient 0402
050A	52		STR R (2)	Résultat stocké à l'adresse 0402
050B	00		IDL	Arrêt

Fig. V. 2. Programme d'addition

III. 1. Mise en service du moniteur

a) Réaliser les connexions entre les modules.

b) Brancher l'alimentation. Le moniteur s'initialise et l'afficheur d'adresse indique PrEt ; sinon, appuyer sur le bouton de remise à zéro de la maquette processeur.

III. 2. Enregistrement du programme à partir de l'adresse 0500 de la RAM.

a) Taper 0500. L'afficheur d'adresse indique 0500. En cas de fausse manœuvre (affichage d'autre chose que 0500), recommencer jusqu'à l'affichage de 0500.

b) Appuyer sur la touche M. L'afficheur de donnée indique le contenu de la ligne d'adresse 0500 de la mémoire.

c) Taper E 2, code de la première instruction. E 2 s'écrit sur l'afficheur de donnée. En cas de fausse manœuvre, recommencer jusqu'à l'affichage de E 2. E 2 est aussi écrit dans la mémoire, à l'adresse 0500.

d) Appuyer sur la touche P. L'afficheur d'adresse indique l'adresse suivante 0501 et l'afficheur de donnée fournit le contenu de la ligne d'adresse 0501.

e) Taper F 8, code de la deuxième instruction. F 8 s'écrit sur l'afficheur de donnée et dans la mémoire, à l'adresse 0501. Recommencer en cas de fausse manœuvre.

f) Appuyer sur la touche P. L'afficheur indique l'adresse suivante 0502 et le contenu de la mémoire à l'adresse 0502.

g) Taper 04 qui s'incrit sur l'afficheur de donnée et dans la mémoire à l'adresse 0502.

h) Reprendre les étapes f) et g) pour toutes les lignes suivantes jusqu'à l'enregistrement de 00 à l'adresse 050B de la mémoire.

i) Appuyer sur la touche I. Le moniteur se réinitialise et indique que le système est *PrEt* à recevoir une adresse.

III. 3. Vérification de l'enregistrement du programme

a) Taper 0500, adresse du début du programme. L'afficheur d'adresse indique 0500 (sinon, fausse manœuvre : retaper 0500 sans se tromper !)

b) Appuyer sur M. Le contenu E 2 de la mémoire à l'adresse 0500 apparaît sur l'afficheur de donnée.

c) Appuyer sur P. L'adresse suivante 0501 apparaît sur l'afficheur ainsi que le contenu F 8 de la mémoire à l'adresse 0501.

d) Reprendre l'étape c) jusqu'à ce que la dernière ligne d'adresse 050B soit vérifiée. En cas d'erreur sur le contenu d'une ligne changer la donnée en agissant sur les touches de caractères hexadécimaux appropriées avant de passer à la vérification de la ligne suivante par action sur la touche P.

e) Appuyer sur la touche I pour réinitialiser le moniteur. L'afficheur indique PrEt.

III. 4. Chargement des données à traiter aux adresses 0400 et 0401.

a) Taper 0400. L'afficheur d'adresse indique 0400.

b) Appuyer sur M. L'afficheur de donnée indique M (0400).

c) Taper la première donnée qui s'écrit sur l'afficheur de donnée et dans la mémoire à l'adresse 0400. On peut introduire n'importe quoi, par exemple 18.

d) Appuyer sur P. L'afficheur d'adresse indique 0401 et l'afficheur de donnée M (0401).

e) Taper la deuxième donnée qui s'écrit sur l'afficheur de donnée et dans la mémoire à l'adresse 0401. Par exemple, on introduira 02.

f) Appuyer sur I. Le moniteur se réinitialise et indique PrEt.

III. 5. Vérification de l'enregistrement des données

Reprendre III. 3. en tapant d'abord l'adresse 0400 et vérifier que la RAM contient bien 18 et 02 aux adresses respectives 0400 et 0401.

III. 6. Exécution du programme

a) Taper 0500, adresse du début du programme. 0500 s'inscrit sur l'afficheur d'adresse.

b) Appuyer sur P. L'afficheur s'éteint et le programme se déroule.

c) Pour réinitialiser le moniteur, il est nécessaire d'actionner le poussoir de remise à zéro du module processeur : le moniteur a en effet perdu le contrôle du système et une pression de la touche I est sans effet.

III. 7. Lecture du résultat

a) Taper 0402, adresse où a dû s'écrire le résultat. L'afficheur indique 0402.

b) Appuyer sur M. L'afficheur de donnée indique 1 A (c'est-à-dire le résultat de l'opération 18 + 02, en hexadécimal, résultat stocké par programme à l'adresse 0402.

IV. Aide à la programmation apportée par le programme moniteur

La mission du programme moniteur n'est pas uniquement de permettre un accès aisé à la mémoire afin d'y effectuer une lecture ou une écriture, d'établir une communication entre le système microprocesseur-mémoire et l'utilisateur en gérant le clavier et l'afficheur, et de permettre le lancement de l'exécution d'un programme. En dehors de cette mission d'ordre « matériel », le programme moniteur peut apporter une aide logicielle à l'utilisateur. Lors de l'établissement d'un programme, l'utilisateur peut, en effet, faire appel à des sous-programmes qui font partie du programme moniteur. Le programme moniteur contient les routines qui permettent la gestion de ces sous-programmes.

Il est prématuré de développer cette question. Attendons les prochains chapitres qui mettront en évidence la structure et les méthodes du programme moniteur.

Cependant pour mettre en lumière cette faculté du programme moniteur, nous proposons de résoudre le problème suivant : réaliser une bascule astable utilisant la sortie Q du microprocesseur.

L'organigramme général de fonctionnement d'une telle bascule est représenté sur la figure V. 3.

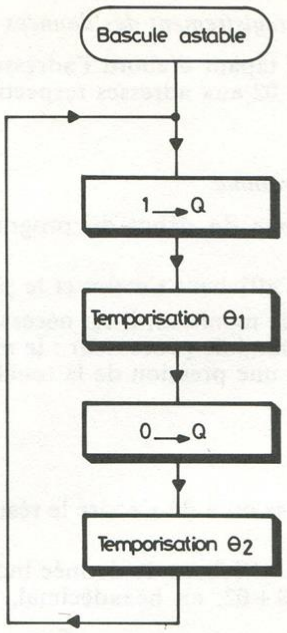


Fig. V. 3. Organigramme d'une bascule astable

Deux possibilités s'offrent au programmeur au niveau de la réalisation des temporisations :

— ou bien il travaille indépendamment du programme moniteur et il aboutit par exemple au programme de la figure V. 4.

— ou bien il demande l'aide du programme moniteur. Il sait alors que celui-ci contient précisément un sous-programme de temporisation auquel on peut faire appel en écrivant le code D 4, puis l'adresse 0100 de ce sous-programme puis deux octets W_1 et W_0 qui précisent la durée de la temporisation (voir explication plus loin). On obtient alors le programme plus court de la figure V. 5.

Nous conseillons au possesseur de la maquette d'écrire ces programmes en mémoire RAM et de les essayer. On peut régler les durées des états où Q = 1 ou 0 en modifiant W_1 et W_0 de chaque temporisation. On montre que $\Theta \simeq (256\ W_1 + W_0)$. 48. T si T est la période de l'horloge (0,5 µs avec le quartz 2 MHz) et W_1 et W_0 en décimal. Attention de convertir W_1 et W_0 en hexadécimal avant chargement dans la mémoire !

Adresse	Code	Opérande	Mnémonique	Commentaire
n	7B		SEQ	1 → Q
n + 1	F8		LDI	
n + 2		$(W_1)_1$		
n + 3	BC		PHI R (C)	
n + 4	F8		LDI	
n + 5		$(W_0)_1$		
n + 6	AC		PLO R (C)	
n + 7	2C		DEC R (C)	Temporisation θ_1 réglée par $(W_1W_0)_1$
n + 8	9C		GHI R (C)	
n + 9	3A		BNZ	
n + A		$(n + 7)_0$		
n + B	2C		DEC R (C)	
n + C	8C		GLO R (C)	
n + D	3A		BNZ	
n + E		$(n + B)_0$		
n + F	7A		REQ	0 → Q
n + 10	F8		LDI	
n + 11		$(W_1)_2$		
n + 12	BC		PHI R (C)	
n + 13	F8		LDI	
n + 14		$(W_0)_2$		
n + 15	AC		PLO R (C)	
n + 16	2C		DEC R (C)	Temporisation θ_2 réglée par $(W_1W_0)_2$
n + 17	9C		GHI R (C)	
n + 18	3A		BNZ	
n + 19		$(n + 16)_0$		
n + 1A	2C		DEC R (C)	
n + 1B	8C		GLO R (C)	
n + 1C	3A		BNZ	
n + 1D		$(n + 1A)_0$		
n + 1E	30		BR	Branchement au début pour recommencer
n + 1F		$(n)_0$		

Fig. V. 4. Réalisation de la bascule sans le moniteur

Adresse	Code	Opérande	Mnémonique	Commentaire
n	7B		SEQ	1 → Q
n + 1	D4		SEP R (4)	Appel de S.P
n + 2		01		Adresse du S.P. de temporisation
n + 3		00		
n + 4		$(W_1)_1$		Réglage de la durée de temporisation θ_1
n + 5		$(W_0)_1$		
n + 6	7A		REQ	0 → Q
n + 7	D4		SEP R (4)	Appel de S.P.
n + 8		01		Adresse du S.P. de temporisation
n + 9		00		
n + A		$(W_1)_2$		Réglage de la durée de temporisation θ_2
n + B		$(W_0)_2$		
n + C	30		BR	Branchement à n (n_0 est l'octet de poids faible de n)
n + D		n_0		

Fig. V. 5. Réalisation à l'aide du programme moniteur

Chapitre 6

L'AFFICHAGE
LA SCRUTATION ET L'ENCODAGE DU CLAVIER

Avant de commencer l'étude générale du moniteur, il est essentiel de bien assimiler la structure matérielle et les techniques de programmation associées utilisées pour l'affichage des adresses et des données, la scrutation et l'encodage du clavier.

Il convient de noter que clavier, afficheurs et circuits associés ont été regroupés sur une maquette autonome afin de permettre leur utilisation éventuelle dans une application propre au lecteur — utilisation qui peut être sans aucun rapport avec celle qui en est faite pour aider à l'enregistrement et à la mise au point d'un programme.

I. Structure matérielle et principes utilisés pour l'affichage

I. 1. Afficheurs et transistors associés

Le dispositif d'affichage est constitué de six afficheurs comportant chacun sept segments (notés *a* à *g*) et un point (d. p.). Les sept segments et le point (fig. VI. 1.) sont des diodes émettrices de lumière (diodes électroluminescentes ou LED) dont les cathodes sont reliées entre elles (afficheur à cathode commune).

Intéressons-nous provisoirement à un de ces afficheurs et supposons que les cathodes des huit LED soient reliées à la masse. Le courant nécessaire pour allumer une LED peut être délivré par un transistor PNP qui permettra une commande (fig. VI. 2). Un niveau logique 1 (tension voisine de 5 V) appliqué sur l'entrée de commande A_n a pour conséquence de bloquer le transistor T_n : la diode est alors « éteinte ». Un niveau logique 0 (tension voisine de 0) appliqué sur A_n sature T_n et permet « d'allumer » la diode ; la résistance R_c permet de limiter le courant à la valeur désirée et la résistance R_B est choisie suffisamment faible pour autoriser la saturation. Le caractère affiché dépend ainsi du code appliqué sur les entrées A_a à A_g et A_{dp} ; par exemple pour afficher la lettre F (sans point), il suffit « d'allumer » les LED a, e, f, g et « d'éteindre » les LED b, c, d, dp et donc d'appliquer le code $(A_a\ A_b\ A_c\ A_d\ A_e\ A_f\ A_g\ A_{dp}) = (01110001)$.

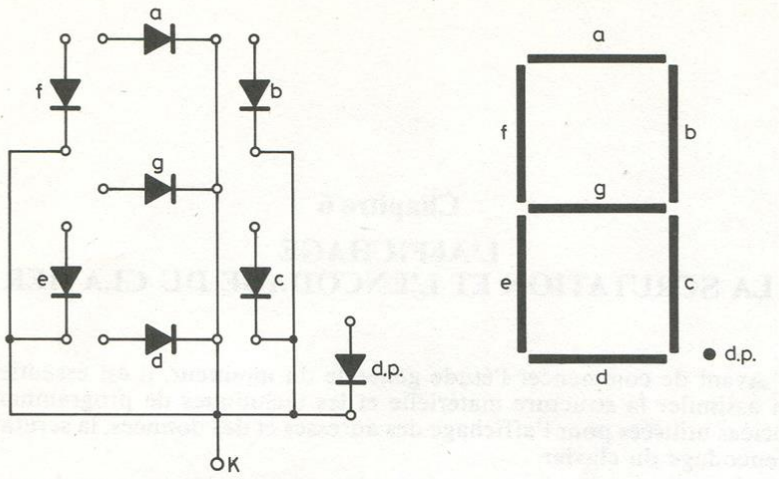


Fig. VI. 1. Constitution d'un afficheur 7 segments à LED

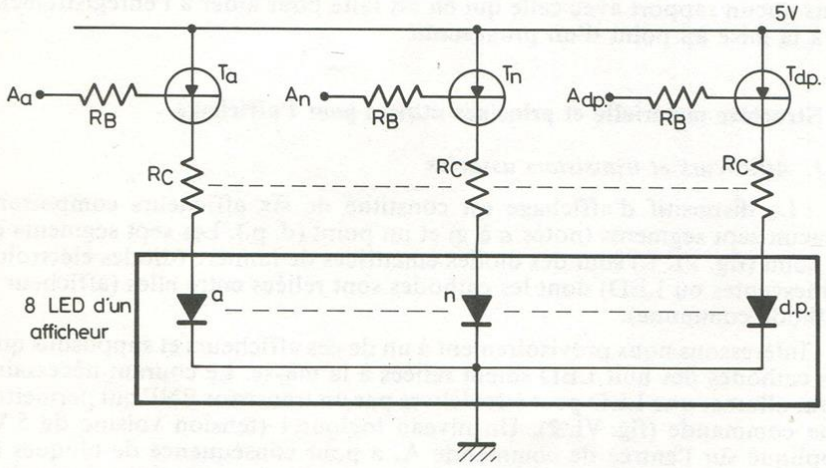


Fig. VI. 2. Commande d'un afficheur

La commande *simultanée* des six afficheurs nécessiterait évidemment six groupes de huit transistors PNP et conduirait à une structure matérielle lourde. Afin de l'éviter, nous avons opté pour une solution de *multiplexage.* La commande des anodes est commune aux six afficheurs (fig. VI. 3.) mais la sélection de l'afficheur concerné est rendue possible grâce à l'introduc-

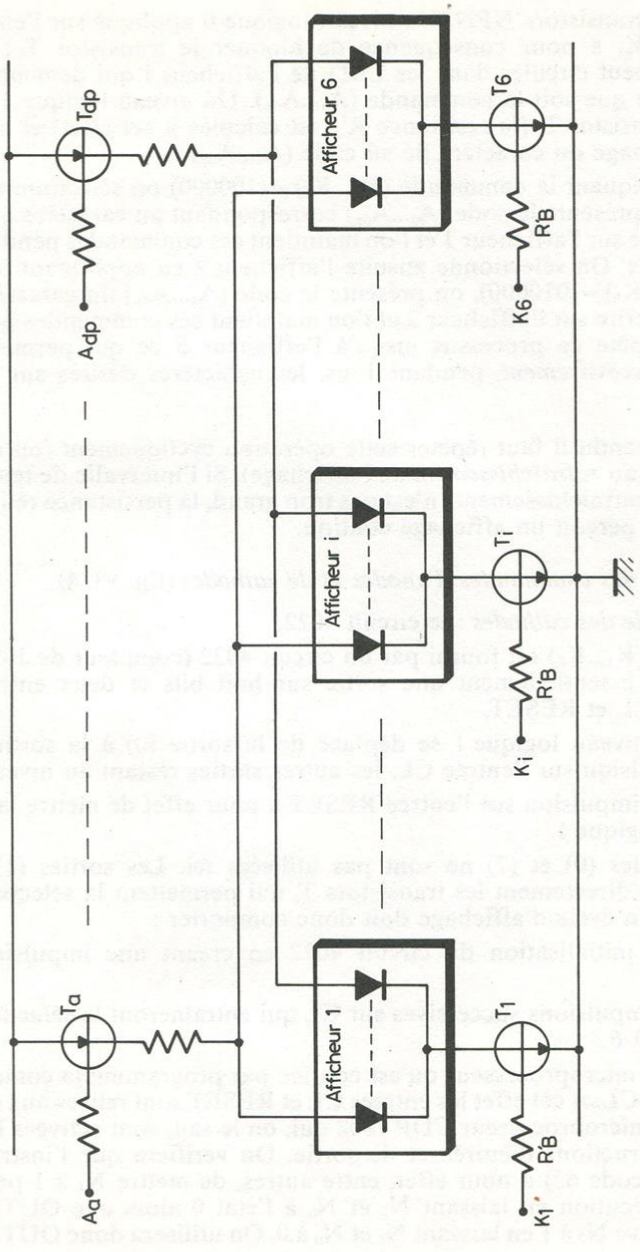


Fig. VI. 3. Sélection des afficheurs

tion des six transistors NPN. Un niveau logique 0 appliqué sur l'entrée de commande K_i a pour conséquence de bloquer le transistor T_i ; aucun courant ne peut circuler dans les LED de l'afficheur i qui demeure donc éteint, quelle que soit la commande $(A_a...A_{dp})$. Un niveau logique 1 sur K_i sature le transistor T_i (la résistance R'_B est calculée à cet effet) et autorise ainsi l'affichage du caractère lié au code $(A_a...A_{dp})$.

En appliquant la commande $(K_1...K_6)=(100000)$ on sélectionne l'afficheur 1 ; on présente le code $(A_a...A_{dp})$ correspondant au caractère que l'on désire inscrire sur l'afficheur 1 et l'on maintient ces commandes pendant par exemple 1 ms. On sélectionne ensuite l'afficheur 2 en appliquant la commande $(K_1...K_6)=(010000)$, on présente le code $(A_a...A_{dp})$ du caractère que l'on veut inscrire sur l'afficheur 2 et l'on maintient ces commandes pendant 1 ms. On répète ce processus jusqu'à l'afficheur 6 ce qui permet donc d'inscrire *successivement,* pendant 1 ms, les caractères désirés sur les six afficheurs.

Bien entendu il faut répéter cette opération cycliquement (on dit que l'on procède au *rafraîchissement* de l'affichage). Si l'intervalle de temps qui sépare deux rafraîchissements n'est pas trop grand, la persistance rétinienne fait que l'on perçoit un affichage continu.

I. 2. Origine des commandes d'anodes et de cathodes (fig. VI. 4)

a) *Commande des cathodes* : le circuit 4022.

Le mot $(K_1...K_6)$ est fourni par un circuit 4022 (compteur de Johnson) qui possède essentiellement une sortie sur huit bits et deux entrées de commande CL et RESET.

— Un niveau logique 1 se déplace de la sortie (0) à la sortie (7) à chaque impulsion sur l'entrée CL, les autres sorties restant au niveau 0.

— Une impulsion sur l'entrée RESET a pour effet de mettre la sortie (0) à l'état logique 1.

Les sorties (0) et (7) ne sont pas utilisées ici. Les sorties (1) à (6) commandent directement les transistors T_i qui permettent la sélection des afficheurs. Un cycle d'affichage doit donc comporter :

— Une initialisation du circuit 4022 en créant une impulsion sur RESET.

— Six impulsions successives sur CL qui entraîneront la sélection des afficheurs 1 à 6.

C'est au microprocesseur qu'est confiée par programme la commande de RESET et CL. A cet effet les entrées CL et RESET sont reliées aux sorties N_1 et N_2 du microprocesseur CDP 1802 qui, on le sait, sont activées lors de certaines instructions d'entrée et de sortie. On vérifiera que l'instruction OUTPUT 2 (code 62) a pour effet, entre autres, de mettre N_1 à 1 pendant le cycle d'exécution en laissant N_2 et N_0 à l'état 0 alors que OUTPUT 4 (code 64) active N_2 à 1 en laissant N_1 et N_0 à 0. On utilisera donc OUTPUT 4

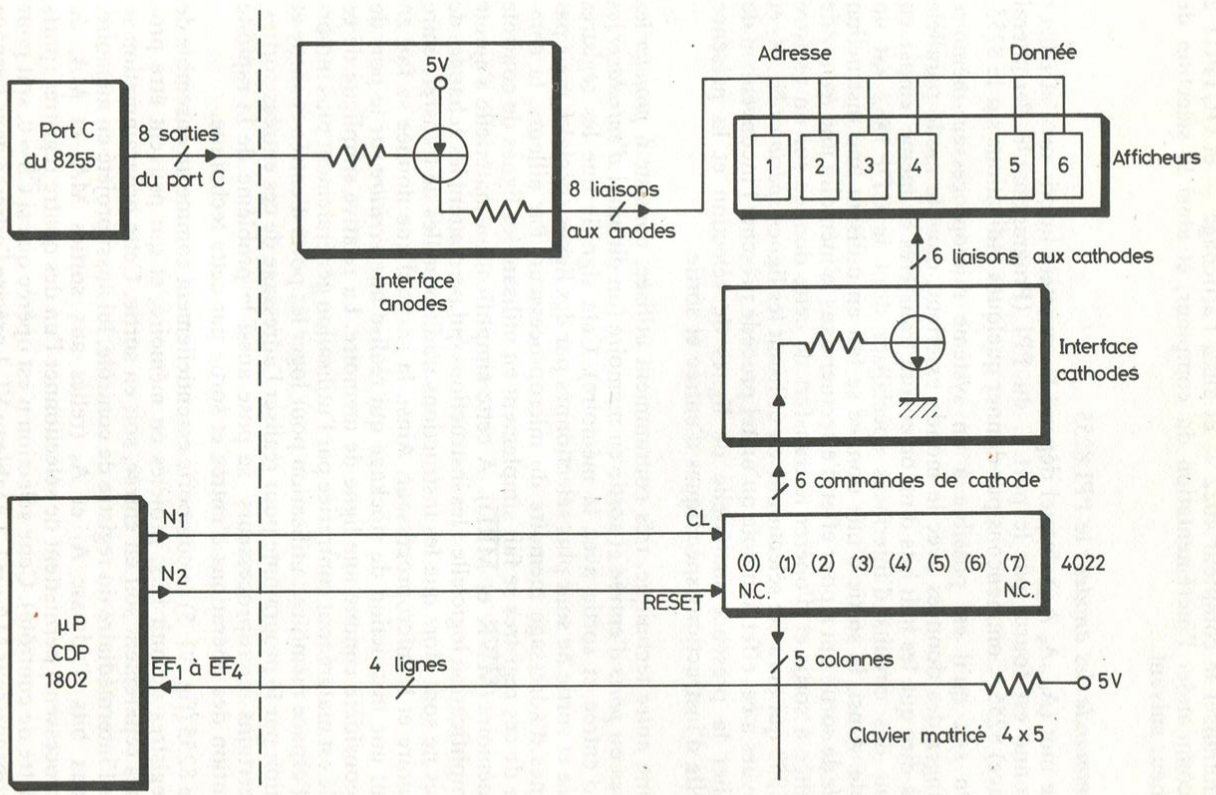


Fig. VI. 4. Commandes des anodes et des cathodes

pour initialiser le compteur 4022 — et aussi l'affichage — et OUTPUT 2 pour commander l'incrémentation du compteur, et ainsi la sélection de l'afficheur suivant.

b) *Commande des anodes :* le PPI 8255

Le mot (A_a...A_g A_{dp}) duquel dépend le caractère inscrit sur l'afficheur sélectionné est fourni par le port C du PPI (Programmable Peripheral Interface) 8255. Commençons par donner quelques indications sur le 8255.

On sait qu'il est possible à un système microprocesseur-mémoire d'échanger des données avec le monde extérieur, dans le mode parallèle (c'est-à-dire que les huit bits d'un octet sont transférés en même temps), en utilisant des circuits d'interfaces spécialisés dont le CDP 1852 est un exemple. Ainsi, la sortie d'une donnée se fait en utilisant une instruction *spéciale* de sortie qui a pour effet d'effectuer dans la mémoire une *lecture* de la donnée à sortir et d'opérer le transfert de cette donnée sur un registre CDP 1852 qui aura été sélectionné en utilisant les lignes *spéciales* N_2, N_1 et N_0 prévues à cet effet. On note qu'un tel procédé présente l'inconvénient de nécessiter la présence matérielle de lignes de sélection et la présence logicielle d'instructions spécifiques d'entrée et sortie.

Une autre technique, très couramment utilisée, consiste à *projeter* les registres ou *ports* d'entrée et sortie en mémoire (on dit aussi *d'entrelacer* les ports d'entrée et sortie avec la mémoire). Cela signifie que les registres d'entrée et sortie ne sont plus sélectionnés par des lignes spéciales mais par les lignes d'adressage mémoire du microprocesseur. Par ailleurs, la commande de ces registres se fait simplement en utilisant les lignes de contrôle de la mémoire ($\overline{MWR}$ et $\overline{MRD}$). A cette simplification matérielle s'ajoute une simplification logicielle : les instructions qui permettent les échanges de données ne sont alors que les instructions traditionnelles de transfert entre la mémoire et le microprocesseur. Ainsi, la sortie d'une donnée se fait en utilisant une instruction de stockage qui réalise une *écriture* sur le port de sortie considéré comme une ligne de mémoire. La relative simplicité de ce procédé est malgré tout contrariée par l'utilisation généralement plus importante d'espace mémoire : utilisation pour loger les ports d'entrée et sortie et utilisation par le programme pour réaliser l'adressage de ces entrées-sorties. Avec certains microprocesseurs se pose aussi le problème de la rapidité d'exécution des opérations d'entrée et sortie par cette technique.

Le 8255 (fig. VI. 5) se comporte essentiellement comme un ensemble de trois registres de huit bits, projetés en mémoire, et qui peuvent être programmés séparément soit en entrée, soit en sortie. Cette programmation se fait par l'intermédiaire du registre de contrôle, lui aussi projeté en mémoire. Les deux bits d'adresse A_1 et A_0 (reliés aux sorties MA_1 et MA_0 de microprocesseur) permettent de sélectionner l'un des quatre registres (ports ou registre de contrôle). Cette sélection n'est opérée que si $\overline{CS} = 0$, soit pour le microprocesseur $MA_{13} = 1$. Le tableau VI. 1 précise l'adresse dans l'espace

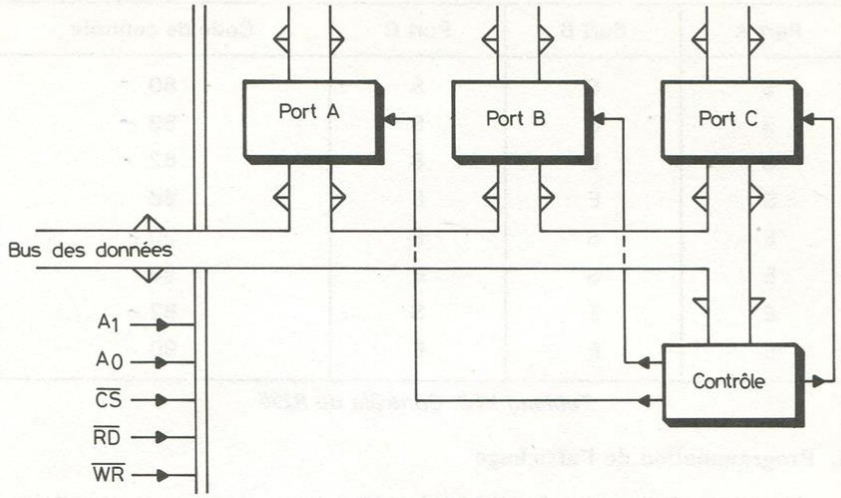


Fig. VI. 5. Schéma fontionnel du 8255

A_1	A_0	Reg. sélectionné	Adresse en hexadécimal
0	0	Port A	2000
0	1	Port B	2001
1	0	Port C	2002
1	1	Reg. de contrôle	2003

Tableau VI-1 Adressage du 8255

adressable par le microprocesseur des quatre registres. Le tableau VI. 2 indique les codes de contrôle des ports A, B et C.

Prenons un exemple. Si l'on désire sortir des données du microprocesseur via le bus des données et le port A ou le port C et entrer des données via le port B, il faut commencer par programmer les ports A et C en sortie et le port B en entrée en stockant le nombre binaire 10000010 (soit 82 en hexadécimal) à l'adresse 2003, c'est-à-dire sur le registre de contrôle. Ceci impose les sens de transfert par les trois ports (fig. VI. 6). Si l'on veut maintenant sortir une donnée via le port C, il suffit d'utiliser l'instruction de stockage de cette donnée à l'adresse 2002.

C'est évidemment au microprocesseur qu'est confiée, par programme, la tâche de fournir les données à sortir par le port C, qui permettront d'inscrire sur les afficheurs les caractères désirés.

Port A	Port B	Port C	Code de contrôle
S	S	S	80
S	S	E	89
S	E	S	82
S	E	E	8B
E	S	S	90
E	S	E	99
E	E	S	92
E	E	E	9B

Tableau VI.2. Contrôle du 8255

II. Programmation de l'affichage

Une zone de la mémoire RAM de 128 octets — que nous appellerons espace d'affichage — est supposée contenir les codes d'affichage (A_a... A_g A_{dp}) correspondant aux caractères que l'on veut écrire sur les afficheurs 1 à 6 (fig.VI. 7).

Par exemple, la ligne d'adresse n+3 contient le code d'affichage CA_4 destiné à l'afficheur 4. En outre la ligne n+6 contient le code FF qui permet l'extinction de tout afficheur sélectionné.

La suite des opérations à réaliser est la suivante :

a) Initialiser le 4022 en envoyant une impulsion sur l'entrée RESET. Ceci est fait en utilisant l'instruction OUTPUT 4.

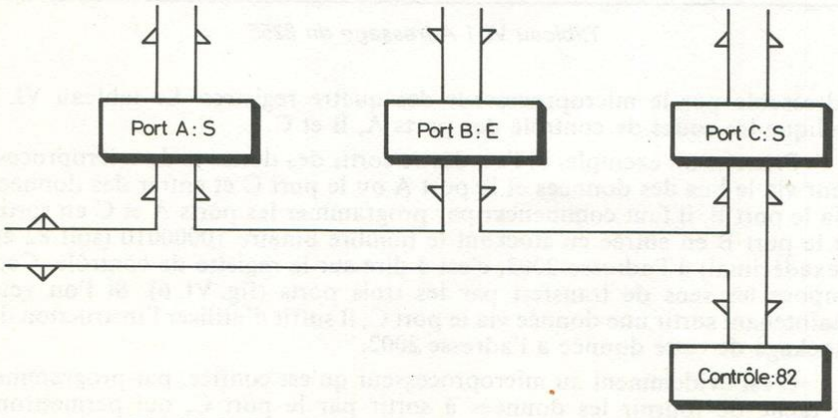


Fig. VI. 6. Commande du sens de transfert des ports

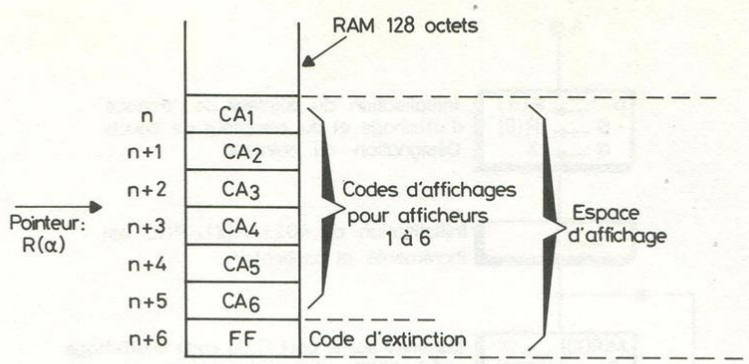


Fig. VI. 7. Espace d'affichage

b) Transférer CA_1 sur le port C.

c) Sélectionner l'afficheur 1 en envoyant une impulsion sur l'entrée CL ce qui est fait en utilisant l'instruction OUTPUT 2. Alors le caractère de code CA_1 s'écrit sur l'afficheur 1.

d) Attendre 1 ms. Ceci est réalisé en utilisant un sous-programme de temporisation.

e) Transférer CA_2 sur le port C.

f) Sélectionner l'afficheur 2 en envoyant une impulsion sur l'entrée CL. Le caractère de code CA_2 s'écrit sur l'afficheur 2 alors que l'afficheur 1 qui n'est plus sélectionné s'éteint.

g) Attendre 1 ms.

h) Reprendre successivement les opérations e, f, g pour les quatre autres afficheurs.

i) Transférer FF sur le port C, ce qui assure qu'aucun afficheur ne s'allumera au cours du programme qui suit cette phase d'affichage.

On remarque que l'on doit effectuer six fois de suite des opérations identiques. Pour éviter la répétition de l'écriture des instructions nécessaires à ces opérations, nous utiliserons la technique de boucle de programme. Le registre R (β) est le compteur de boucles.

L'organigramme du programme d'affichage apparaît sur la figure VI. 8. Il suppose que le registre R (γ) contient l'adresse 2002 du port C. On retrouvera dans le programme moniteur (fin du chapitre 9) le sous-programme d'affichage écrit en langage assembleur et en langage machine à partir de l'adresse 010D (on a alors choisi R (α) = R (F), R (β) = R (9), R (γ) = R (7).

Rappelons qu'il est nécessaire de *rafraîchir* l'affichage périodiquement pour que l'œil ait l'impression d'une présence continue des caractères sur les

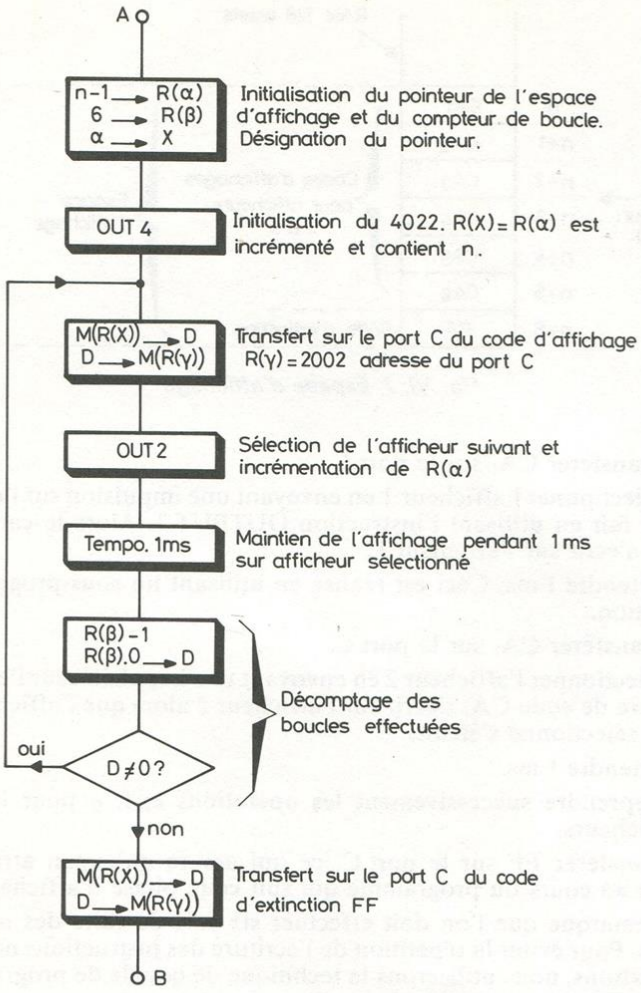


Fig. VI. 8. Organigramme du programme d'affichage

afficheurs. Ceci peut être réalisé simplement en rebouclant le programme compris entre les points A et B sur lui-même (fig. VI. 9.). Cependant, lors de la réalisation d'un projet, le microprocesseur est chargé d'autres missions que la gestion de l'affichage — l'une de ces missions peut être la modification du contenu de l'espace d'affichage. Ces missions devront toutes se terminer par un rafraîchissement (fig. VI. 10).

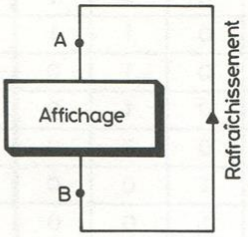


Fig. VI. 9. Bouclage du programme

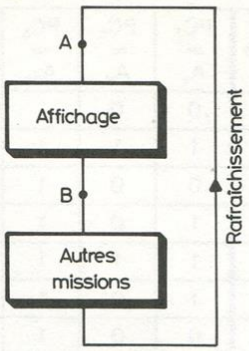


Fig. VI. 10. Bouclage avec d'autres missions intercalées

Remarque

Pour afficher un caractère sur l'afficheur n, il faut connaître son code CA_n. L'établissement de ce code suppose que l'on sache comment sont réalisées les connexions entre les sorties PC_7 à PC_0 du port C et les entrées de commandes A_a à A_{dp} des anodes des afficheurs. Le tableau VI. 3 indique les correspondances réalisées sur la maquette et montre comment sont construits les codes d'affichage des caractères hexadécimaux, ainsi que des lettres P, r, et t utilisées dans le moniteur. Le lecteur peut compléter ce tableau pour trouver les codes des caractères qu'il a personnellement besoin d'afficher. La figure VI. 11 rappelle la représentation des caractères hexadécimaux en sept segments (attention de ne pas confondre 6 et B).

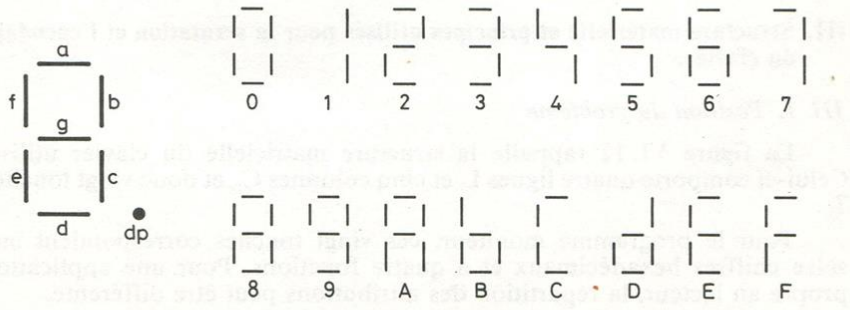


Fig. VI. 11. Représentation des caractères hexadécimaux

Caractère	PC_7 = A_e	PC_6 = A_d	PC_5 = A_{dp}	PC_4 = A_c	PC_3 = A_a	PC_2 = A_b	PC_1 = A_f	PC_0 = A_g	Code d'affichage
0	0	0	1	0	0	0	0	1	21
1	1	1	1	0	1	0	1	1	EB
2	0	0	1	1	0	0	1	0	32
3	1	0	1	0	0	0	1	0	A2
4	1	1	1	0	1	0	0	0	E8
5	1	0	1	0	0	1	0	0	A4
6	0	0	1	0	0	1	0	0	24
7	1	1	1	0	0	0	1	1	E3
8	0	0	1	0	0	0	0	0	20
9	1	1	1	0	0	0	0	0	E0
A	0	1	1	0	0	0	0	0	60
B	0	0	1	0	1	1	0	0	2C
C	0	0	1	1	0	1	0	1	35
D	0	0	1	0	1	0	1	0	2A
E	0	0	1	1	0	1	0	0	34
F	0	1	1	1	0	1	0	0	74
P	0	1	1	1	0	0	0	0	70
r	0	1	1	1	1	1	1	0	7E
t	0	0	1	1	1	1	0	0	3C

Tableau VI.3. Codes

III. Structure matérielle et principes utilisés pour la scrutation et l'encodage du clavier.

III. 1. Position du problème

La figure VI. 12 rappelle la structure matricielle du clavier utilisé. Celui-ci comporte quatre lignes L_i et cinq colonnes C_j, et donc vingt touches T_{ij}.

Pour le programme moniteur, ces vingt touches correspondent aux seize chiffres hexadécimaux et à quatre fonctions. Pour une application propre au lecteur, la répartition des attributions peut être différente.

Appuyer sur une touche du clavier, c'est toujours demander au

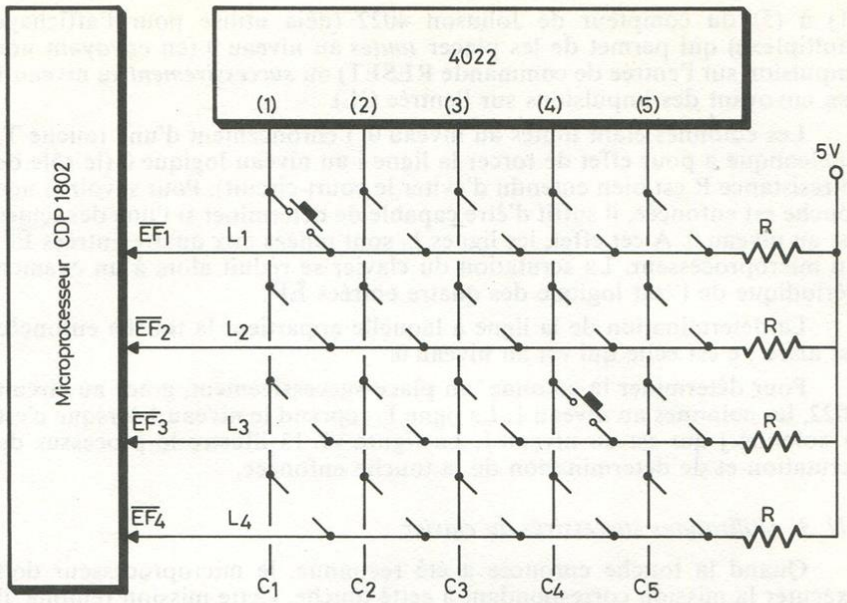


Fig. VI. 12. Le clavier utilisé

microprocesseur d'effectuer une certaine mission. Le problème de la gestion du clavier se pose en ces termes :

— le manipulateur a-t-il demandé par l'intermédiaire du clavier l'exécution d'une mission particulière ? Autrement dit, une touche du clavier a-t-elle été enfoncée ? La réponse à cette question ne peut être donnée que s'il existe une procédure d'examen périodique de l'état du clavier appelée *scrutation*, opération à laquelle le microprocesseur apporte évidemment son concours ;

— si une mission particulière est demandée, quelle est cette mission ? C'est-à-dire quelle est la touche qui est enfoncée ? Pour le savoir, on va s'aider du microprocesseur pour construire un mot binaire qui sera caractéristique de la touche enfoncée. C'est l'opération d'*encodage* du clavier. La lecture du code généré permettra ultérieurement de savoir quelle est la touche qui a été enfoncée et donc de connaître la mission à accomplir.

III. 2. Conception matérielle

Pour que la résolution de ce double problème soit matériellement possible, les conducteurs de lignes sont portés au potentiel 5 V (c'est-à-dire au niveau logique 1) par l'intermédiaire des résistances R, lorsqu'aucune touche n'est enfoncée. Les conducteurs de colonnes sont reliés aux sorties

(1) à (5) du compteur de Johnson 4022 (déjà utilisé pour l'affichage multiplexé) qui permet de les placer *toutes* au niveau 0 (en envoyant une impulsion sur l'entrée de commande RESET) ou *successivement* au niveau 1 (en envoyant des impulsions sur l'entrée CL).

Les colonnes étant toutes au niveau 0, l'enfoncement d'une touche T_{ij} quelconque a pour effet de forcer la ligne i au niveau logique 0 (le rôle de la résistance R est bien entendu d'éviter le court-circuit). Pour savoir si une touche est enfoncée, il suffit d'être capable de déterminer si l'une des lignes est au niveau 0. A cet effet, les lignes L_i sont reliées aux quatre entrées $\overline{EF}_i$ du microprocesseur. La scrutation du clavier se réduit alors à un examen périodique de l'état logique des quatre entrées $\overline{EF}_i$.

La détermination de la ligne à laquelle appartient la touche enfoncée est aisée : c'est celle qui est au niveau 0.

Pour déterminer la colonne, on place successivement, grâce au circuit 4022, les colonnes au niveau 1. La ligne L_i reprend le niveau 1 lorsque c'est la colonne j qui est au niveau 1. La figure VI. 13 illustre le processus de scrutation et de détermination de la touche enfoncée.

III. 3. Utilisations successives du clavier

Quand la touche enfoncée a été reconnue, le microprocesseur doit exécuter la mission correspondant à cette touche. Cette mission remplie, il faut généralement reprendre la scrutation du clavier pour savoir si une autre mission sera requise. On aboutit ainsi à l'organigramme général de la figure VI. 14 qui présente une structure de boucle et permet la prise en compte de requêtes successives.

Il convient cependant de remarquer que encodage de la touche pressée et exécution de la mission peuvent être très rapides et il est fort possible que l'on revienne à la scrutation du clavier alors que la touche n'est pas relâchée ce qui est interprété, sûrement à tort, comme une nouvelle requête de la même mission. Ainsi, la mission demandée risque de s'exécuter un nombre aléatoire de fois. La solution à ce problème consiste à inclure dans la boucle un test de relâchement de la touche, par exemple après l'opération d'encodage. La scrutation du clavier ne peut alors reprendre que si la touche est relâchée (fig. VI. 15).

III. 4. Problème du rebondissement

L'enfoncement de la touche T_{ij} n'a malheureusement pas pour seul effet d'imposer un niveau 0 sur la ligne L_i. L'élasticité du ressort de rappel de la touche a pour conséquence son rebondissement sur le contact de fermeture. Une succession de 0 et de 1 qui apparaissent sur la ligne L_i peuvent être interprétés comme autant de pressions et relâchements successifs de la touche T_{ij} et peuvent entraîner l'exécution multiple de la mission correspondante. On vérifie expérimentalement que les rebonds sont inexistants au bout de 20 ms.

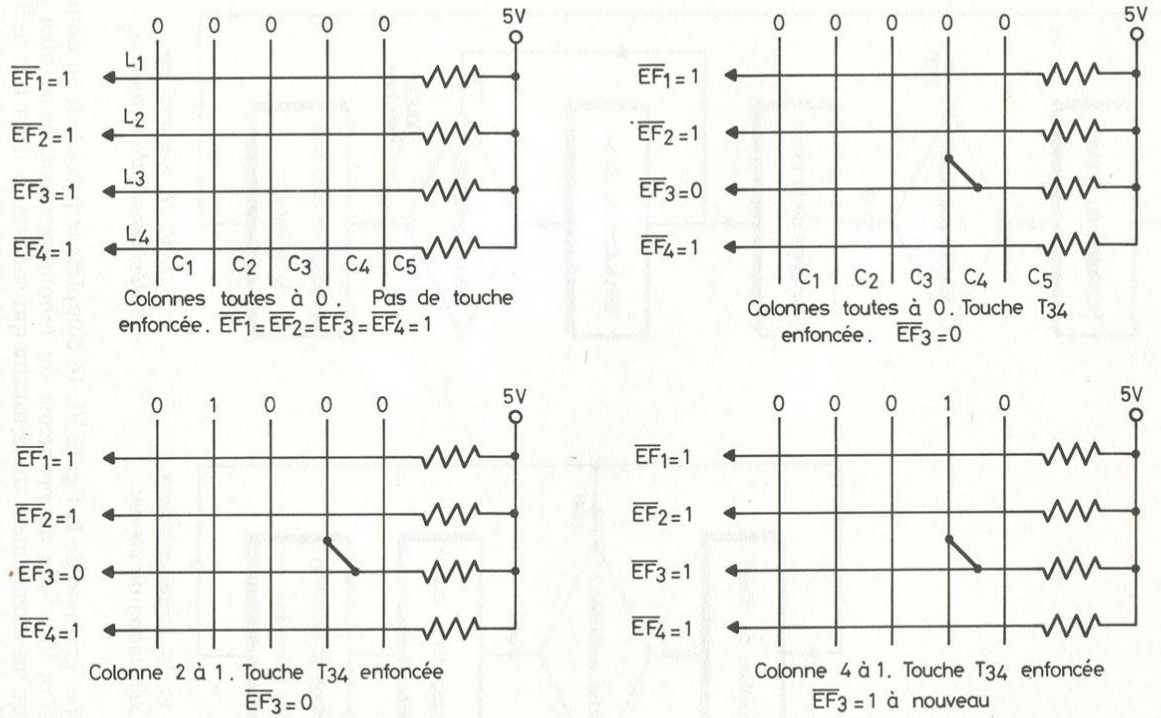


Fig. VI. 13. Processus de scrutation du clavier

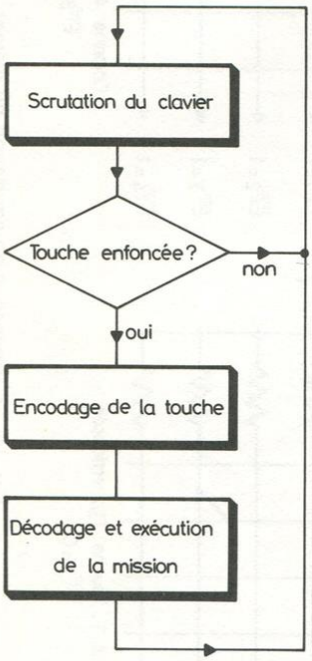


Fig. VI. 14. Organigramme de scrutation du clavier

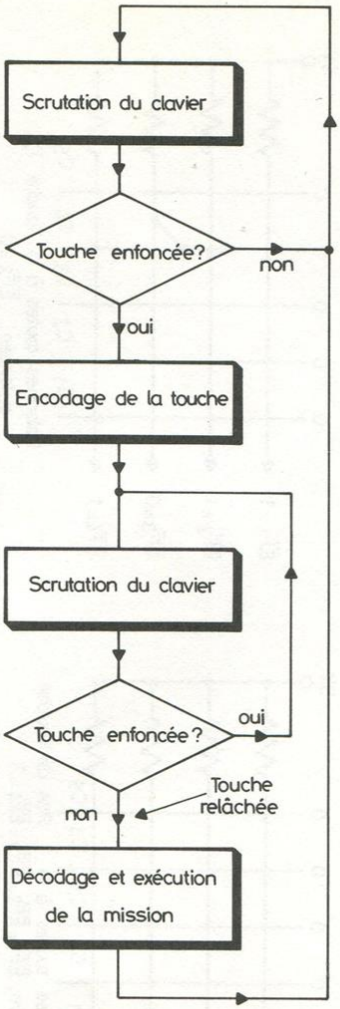


Fig. VI. 15. Sécurité de relâchement de la touche

L'organigramme de la figure VI. 16 complète le précédent en donnant une solution logicielle au problème du rebondissement, aussi bien à la pression qu'au relâchement de la touche qui sont testés deux fois à 20 ms d'intervalle. Cette solution assure en plus une immunité aux bruits de faible durée.

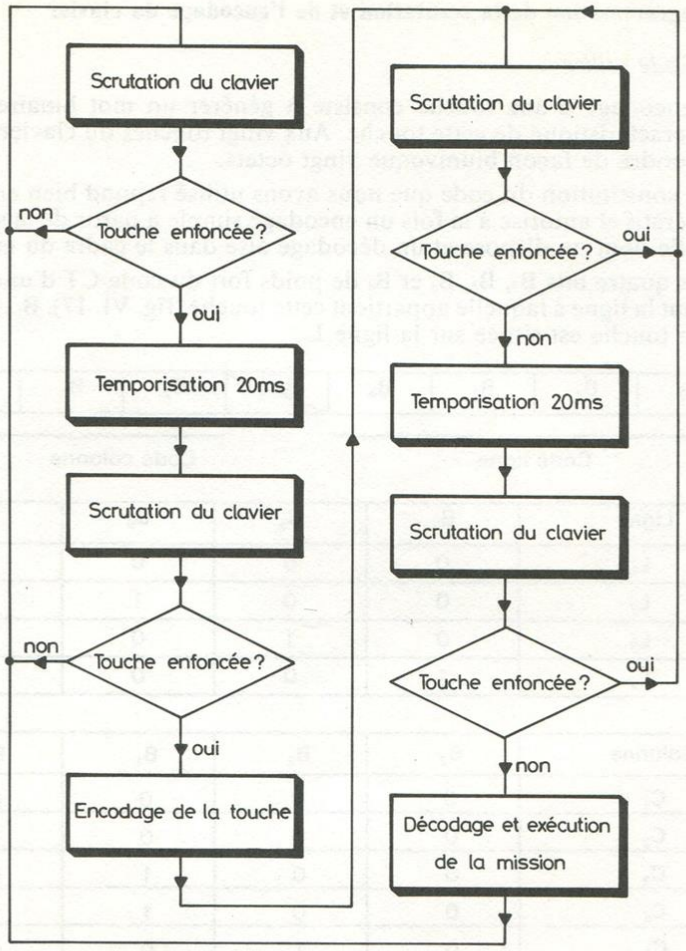


Fig. VI. 16. Suppression des « rebondissements »

Remarque :

On pourrait penser que l'utilisation d'un clavier à touches sensitives apporte au problème du rebondissement une solution matérielle (pas de ressorts de rappel de touches). Hélas, le rebondissement est remplacé par le tremblement, même léger, du manipulateur ! Les conséquences sont exactement les mêmes... le remède logiciel aussi. Le maniement du clavier à touches sensitives exige des « pressions » franches, le doigt se posant bien à plat.

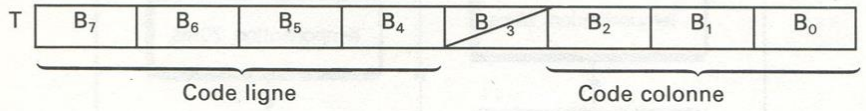
IV. Programmation de la scrutation et de l'encodage du clavier

IV. 1. Code utilisé

L'encodage d'une touche consiste à générer un mot binaire (ici un octet) caractéristique de cette touche. Aux vingt touches du clavier doivent correspondre de façon biunivoque vingt octets.

La constitution du code que nous avons utilisé répond bien entendu à cet impératif et autorise à la fois un encodage simple à partir de la structure matérielle dont on dispose et un décodage aisé dans le cadre du moniteur.

Les quatre bits B_7, B_6, B_5 et B_4 de poids fort du code CT d'une touche indiquent la ligne à laquelle appartient cette touche (fig. VI. 17). B_{i+3} est seul à 1 si la touche est située sur la ligne L_i.



Ligne	B_7	B_6	B_5	B_4
L_1	0	0	0	1
L_2	0	0	1	0
L_3	0	1	0	0
L_4	1	0	0	0

Colonne	B_3	B_2	B_1	B_0
C_1	0	0	0	0
C_2	0	0	0	1
C_3	0	0	1	0
C_4	0	0	1	1
C_5	0	1	0	0

Fig. VI. 17. Code des touches

Il n'est pas possible d'utiliser une codification aussi simple pour les cinq colonnes car on ne dispose plus que de quatre bits. Convenons d'utiliser la numération binaire naturelle. L'utilisation des seuls bits B_2, B_1, B_0 est suffisante : le bit B_3 n'est pas utilisé. A la colonne 1 correspond le code $(B_2\ B_1\ B_0) = (000)$, à la colonne 2 est associé $(B_2\ B_1\ B_0) = (001)$... On note que $B_2 = 1$ *uniquement* pour une touche de la colonne 5 (cette colonne est reservée aux fonctions dans le cadre du moniteur).

IV. 2. Scrutation du clavier

L'organigramme de scrutation du clavier est donné sur la figure VI. 18. Il consiste essentiellement à tester successivement l'état logique des entrées

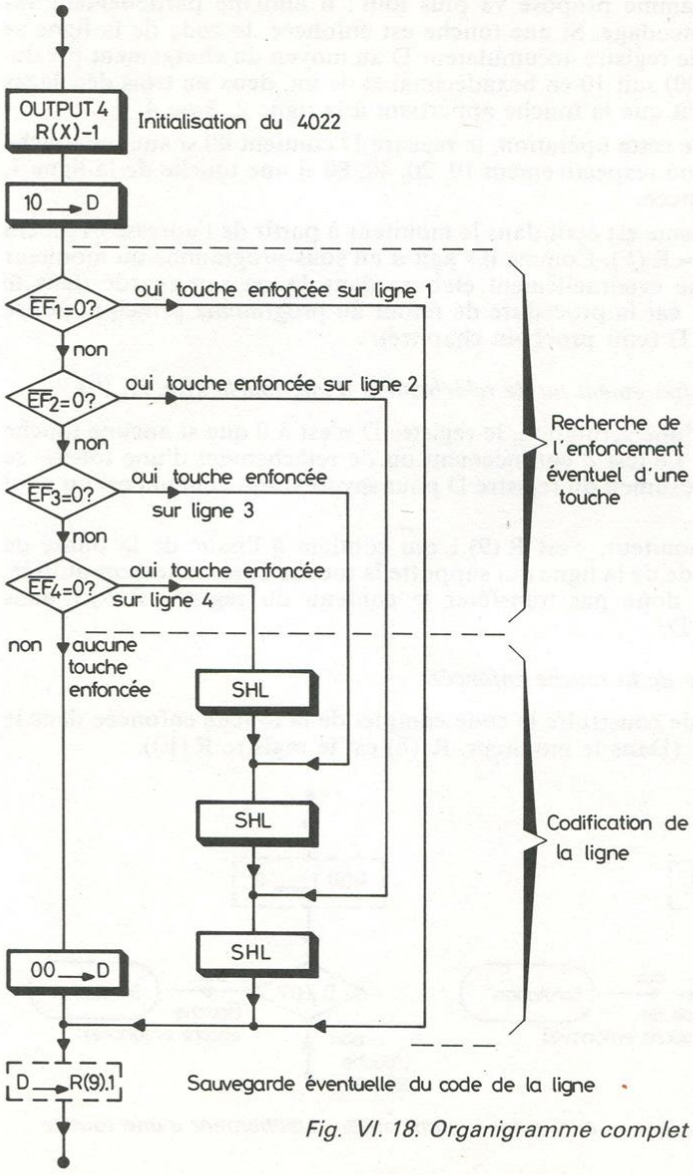


Fig. VI. 18. Organigramme complet de la scrutation

$\overline{EF_i}$. Si aucune touche n'est enfoncée (toutes les entrées $\overline{EF_i}$ sont à 1), le registre accumulateur D est mis à 0. Si une touche est enfoncée, on aboutit à $D \neq 0$.

L'organigramme proposé va plus loin : il anticipe partiellement sur l'opération d'encodage. Si une touche est enfoncée, le code de la ligne se construit dans le registre accumulateur D au moyen du chargement préalable de (00010000) soit 10 en hexadécimal et de un, deux ou trois décalages à gauche suivant que la touche appartient à la ligne 2, 3 ou 4.

A l'issue de cette opération, le registre D contient 00 si aucune touche n'est enfoncée ou respectivement 10, 20, 40, 80 si une touche de la ligne 1, 2, 3, 4 est enfoncée.

Le programme est écrit dans le moniteur à partir de l'adresse 0125. On a encore R (X) = R (F). Comme il s'agit d'un sous-programme du moniteur le code colonne éventuellement élaboré dans D est sauvegardé dans le registre R (9). 1 car la procédure de retour au programme principal utilise l'accumulateur D (voir prochain chapitre).

IV. 3. Test d'enfoncement ou de relâchement d'une touche (fig.VI. 19)

A l'issue d'une scrutation, le registre D n'est à 0 que si aucune touche n'est enfoncée. Le test d'enfoncement ou de relâchement d'une touche se réduit donc à l'examen du registre D pour savoir si son contenu est ou n'est pas 0.

Dans le moniteur, c'est R (9).1 qui contient à l'issue de la phase de scrutation le code de la ligne qui supporte la touche éventuellement utilisée. On commence donc par transférer le contenu du registre R (9).1 dans l'accumulateur D.

IV. 4. Encodage de la touche enfoncée

Décidons de construire le code complet de la touche enfoncée dans le registre R (δ).0. (Dans le moniteur, R (δ) est le registre R (E)).

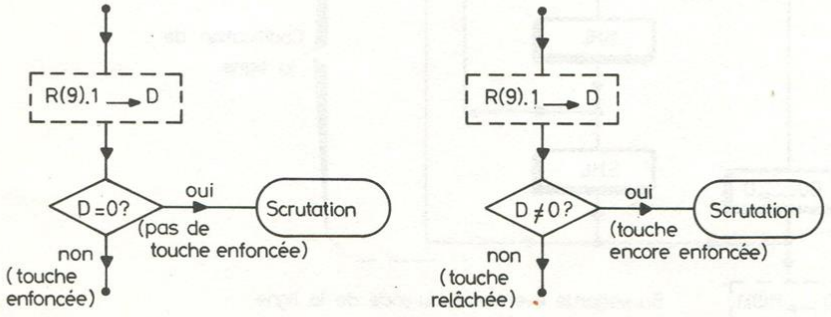


Fig. VI. 19. Détection de l'enfoncement ou du relâchement d'une touche

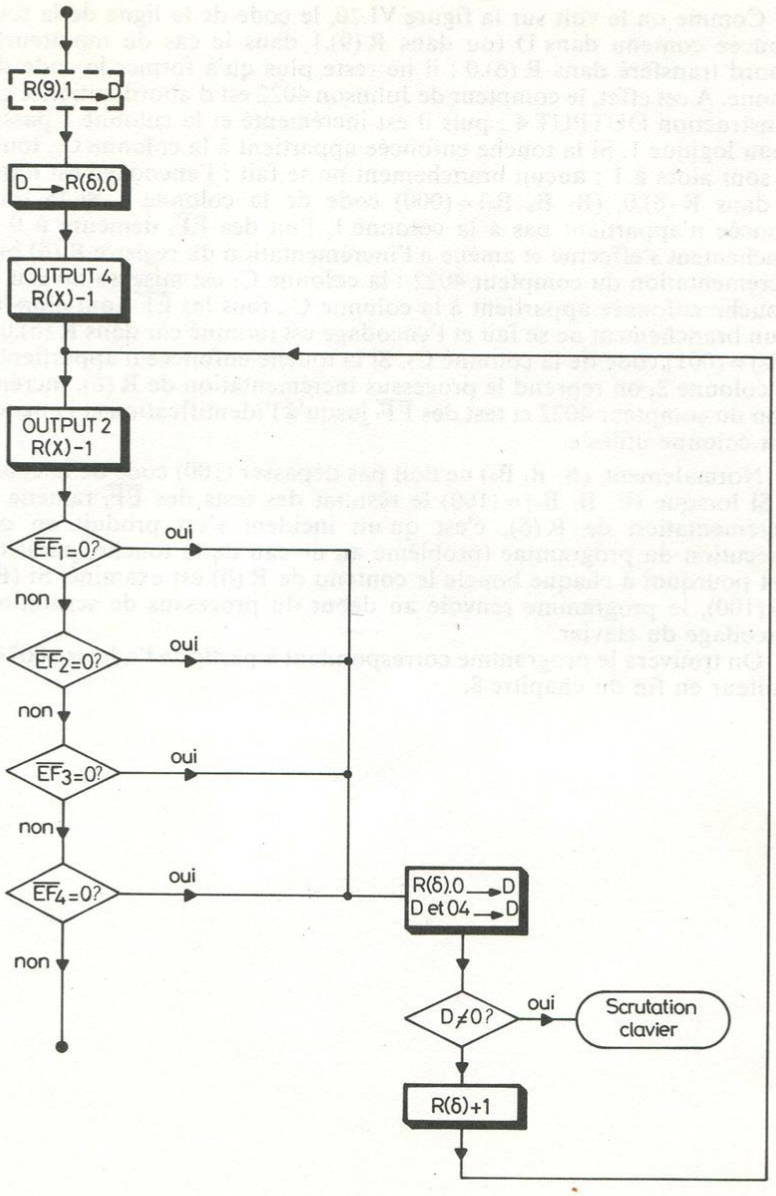
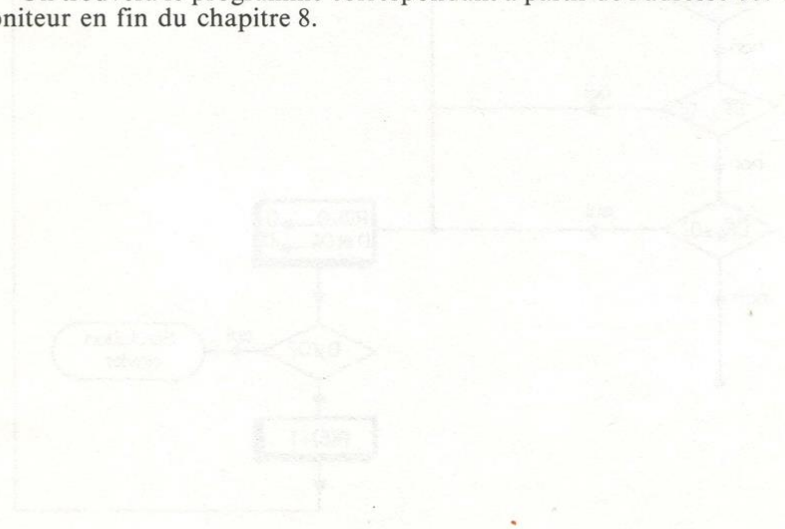


Fig. VI.20. Organigramme du programme d'encodage de la touche enfoncée

Comme on le voit sur la figure VI.20, le code de la ligne de la touche enfoncée contenu dans D (ou dans R (9).1 dans le cas du moniteur) est d'abord transféré dans R (δ).0 : il ne reste plus qu'à former le code de la colonne. A cet effet, le compteur de Johnson 4022 est d'abord initialisé grâce à l'instruction OUTPUT 4 ; puis il est incrémenté et la colonne 1 passe au niveau logique 1. Si la touche enfoncée appartient à la colonne C_1, tous les $\overline{EF_i}$ sont alors à 1 ; aucun branchement ne se fait ; l'encodage est terminé car dans R (δ).0, (B_2 B_1 B_0)=(000) code de la colonne 1. Si la touche enfoncée n'appartient pas à la colonne 1, l'un des $\overline{EF_i}$ demeure à 0 ; un branchement s'effectue et amène à l'incrémentation du registre R (δ) puis à l'incrémentation du compteur 4022 : la colonne C_2 est mise au niveau 1. Si la touche enfoncée appartient à la colonne C_2, tous les $\overline{EF_i}$ sont alors à 1 ; aucun branchement ne se fait et l'encodage est terminé car dans R (δ).0, (B_2 B_1 B_0)=(001) code de la colonne C_2. Si la touche enfoncée n'appartient pas à la colonne 2, on reprend le processus incrémentation de R (δ), incrémentation du compteur 4022 et test des $\overline{EF_i}$ jusqu'à l'identification et l'encodage de la colonne utilisée.

Normalement, (B_2 B_1 B_0) ne doit pas dépasser (100) code de la colonne C_5. Si lorsque (B_2 B_1 B_0)=(100) le résultat des tests des $\overline{EF_i}$ ramène vers l'incrémentation de R (δ), c'est qu'un incident s'est produit en cours d'exécution du programme (problème au niveau de la touche, parasite...). C'est pourquoi à chaque boucle le contenu de R (δ) est examiné. Si (B_2 B_1 B_0=(100), le programme renvoie au début du processus de scrutation et d'encodage du clavier.

On trouvera le programme correspondant à partir de l'adresse 0074 du moniteur en fin du chapitre 8.



Chapitre 7
TECHNIQUES DES SOUS-PROGRAMMES

Avant de procéder à la description du programme moniteur dans son ensemble, il est indispensable d'exposer les méthodes de gestion de sous-programmes qu'autorisent l'architecture et le jeu d'instruction du microprocesseur CDP 1802.

Le lecteur ne doit pas comprendre ce chapitre seulement comme une préparation à l'explication du moniteur mais plutôt comme une formation à des techniques qui lui seront utiles pour la réalisation de ses propres programmes. Dans cette optique, le programme moniteur apparaîtra comme un exemple d'utilisation de ces méthodes.

I. Les sous-programmes

Dans les programmes importants, il arrive souvent qu'une séquence d'instructions soit utilisée à beaucoup de reprises. Par exemple, le moniteur et ses programmes complémentaires comportent de nombreuses temporisations.

Une solution directe pour programmer ces temporisations est d'insérer la séquence convenable d'instructions à chaque endroit du programme où l'on a besoin de cette fonction. On conçoit facilement que cette duplication de la même séquence, en dehors même de son aspect fastidieux pour celui qui a à écrire le programme, consomme beaucoup de lignes de la mémoire, d'autant plus que la séquence est plus longue et répétée plus souvent.

Une autre solution consiste à n'écrire la séquence qu'*une seule fois* et à l'utiliser chaque fois que l'on en a besoin. Cet usage s'effectue en écrivant la fonction sous forme d'un *sous-programme* auquel le programme en cours peut faire *appel* grâce à une instruction particulière engendrant un mécanisme à définir. Quand le sous-programme a accompli sa fonction, il réalise le *retour* automatique au programme qui l'a appelé, suivant une procédure qui reste aussi à définir.

Signalons que la plupart des microprocesseurs sont microprogrammés pour permettre, en utilisant des instructions spécifiques, l'appel d'un sous-programme résidant à une adresse déterminée et le retour au programme appelant.

Si le CDP 1802 ne possède pas de telles instructions d'appel et de retour, sa structure logicielle permet d'en concevoir des équivalences. Nous nous proposons d'exposer ici deux techniques, la première très simple et par voie de conséquence d'usage un peu limité, la seconde plus élaborée mais aussi de caractère universel.

II. La technique n'utilisant que le changement de compteur de programme

II. 1. Changement de registre compteur de programme

On sait que, après initialisation, le déroulement d'un programme commence toujours à l'adresse 0000 de la mémoire, le registre compteur de programme étant R (0). Pour des raisons diverses (libération de R (0) pour utilisation éventuelle comme pointeur en DMA, gestion de sous-programmes...) il peut être nécessaire d'attribuer le rôle de compteur de programme à un autre registre. Le changement de compteur de programme se fait simplement en modifiant par programme le contenu du registre P au moyen de l'instruction SET P.

Supposons que, à la ligne d'adresse Z de la mémoire pointée par le registre compteur de programme R (α) on ait inscrit le code Dβ de l'opération $\beta \rightarrow$ P. Lors du cycle de *recherche* de cette instruction, le code Dβ est chargé dans les registres d'instruction I et N du microprocesseur et R (α) est incrémenté : son contenu devient Z + 1. Pendant le cycle d'*exécution*, β est chargé dans le registre P. A l'issue de ce cycle, *R (β) est compteur de programme*. Ceci signifie que, pendant le cycle de recherche de l'instruction suivante, c'est le contenu de la ligne d'adresse W écrite dans le registre R (β) qui va être chargé dans le registre d'instruction du microprocesseur et le programme se poursuivra donc à partir de l'adresse W. Bien entendu le contenu W du registre R (β) aura été préalablement fixé par programme (fig. VII. 1). Une telle opération réalise une rupture de séquence (il y a « saut » de programme de l'adresse Z à l'adresse W) ; elle se distingue d'un branchement par la modification du compteur de programme qu'elle engendre.

On trouvera, dans le programme moniteur, un exemple de changement de compteur de programme : les initialisations générales de registres sont effectuées avec R (0) comme compteur puis on affecte à R (3) ce rôle grâce à l'instruction de mnémonique SEP R (3) (ligne 0019) ; le programme se poursuit à l'adresse 0037 *préalablement chargée* dans R (3).

II. 2. Appel d'un sous-programme et retour au programme appelant

Dans cette technique, un registre particulier est affecté pour servir de compteur de programme propre à chaque sous-programme. C'est le processus que l'on vient de décrire qui est utilisé pour faire appel à un sous-programme et, de ce sous-programme, revenir au programme appelant.

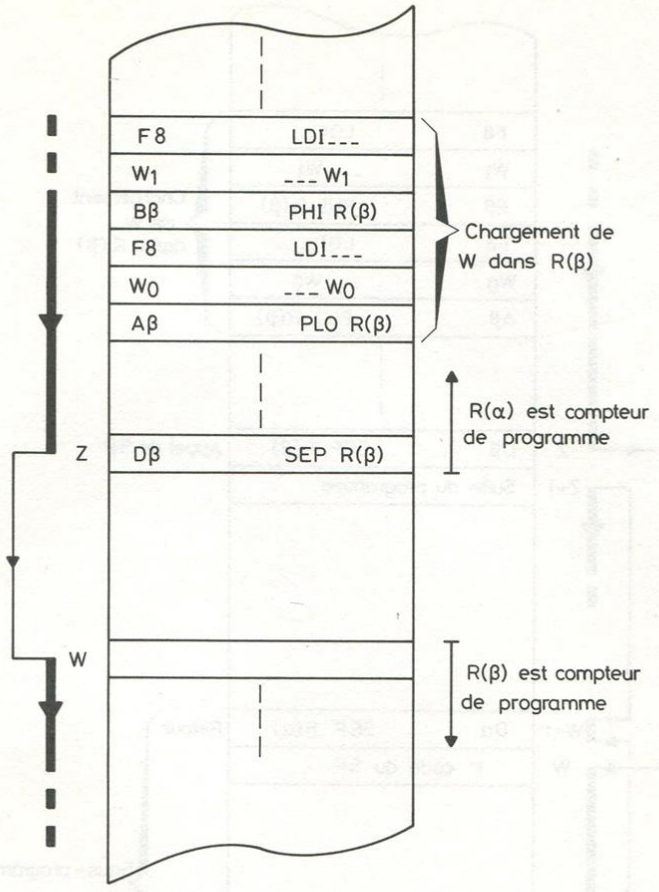


Fig. VII. 1. Changement de compteur de programme : R (β) remplace R (α)

Reportons-nous à la figure VII. 2. Un programme est décrit en utilisant le registre R (α) comme compteur de programme. Un sous-programme est enregistré à partir de l'adresse W ; R (β) en sera le compteur de programme. Le registre R (β) ayant été initialisé à W, *l'appel du sous-programme* est effectué par simple modification du compteur de programme grâce à l'instruction SEP R (β) dont le code est écrit à l'adresse Z. Quand le sous-programme a rempli sa fonction, *le retour* au programme appelant se fait tout simplement en programmant l'instruction SEP R (α) qui a pour effet de confier à nouveau à R (α) le rôle de compteur de programme. Comme le registre R (α) a été abandonné avec un contenu Z + 1 lors de

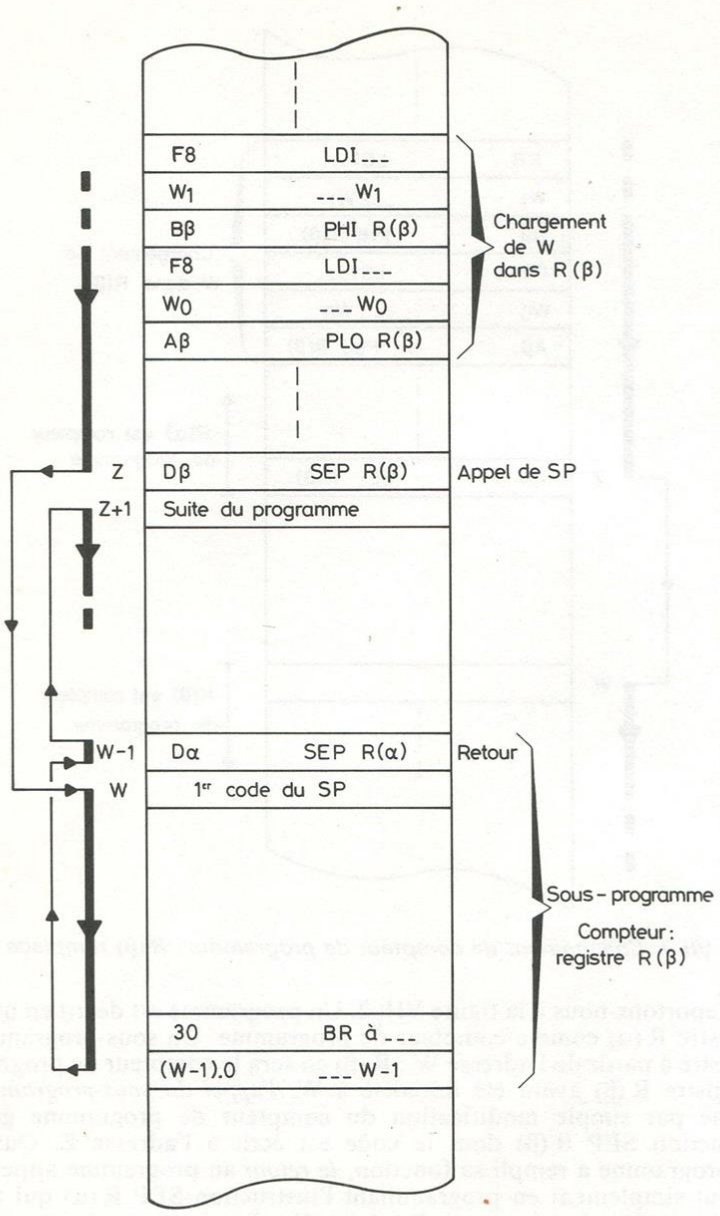


Fig. VII. 2. Appel d'un sous-programme et retour par changement de compteur de programme

l'appel du sous-programme, le programme appelant continue de se dérouler à partir de cette adresse. Il est astucieux de faire en sorte (grâce à un branchement au cours du sous-programme) que l'instruction SEP R (α) de retour au programme appelant soit écrite à l'adresse W – 1. Ainsi, on quitte le sous-programme en laissant le registre R (β) avec le contenu W, ce qui en prépare un appel ultérieur (son compteur de programme est initialisé automatiquement).

II. 3. Avantages et inconvénients de cette technique

L'intérêt de cette technique réside essentiellement dans sa simplicité de mise en œuvre et dans la rapidité de l'exécution de l'appel et du retour qui ne nécessitent qu'un cycle machine.

Une première limitation d'utilisation se trouve dans le fait que *chaque* sous-programme requiert son propre compteur de programme (à moins de prévoir une procédure d'initialisation du compteur avant chaque appel, ce qui est lourd, fastidieux et demande du temps d'exécution) : le nombre des sous-programmes ne pourra pas dépasser la dizaine (on ne dispose que de 16 registres en tout).

Par ailleurs, tous les sous-programmes sont mutuellement exclusifs en ceci qu'il est interdit qu'un sous-programme d'un programme principal appelle un autre sous-programme de ce même programme principal. Supposons en effet qu'un programme principal de compteur R (α) puisse appeler deux sous-programmes SP_β et SP_γ de compteurs respectifs R (β) et R (γ). Chacun de ces deux sous-programmes se termine par l'instruction SEP R (α) de retour au programme principal. S'il semble possible d'appeler à partir de SP_β par exemple le sous-programme SP_γ, le retour à SP_β ne peut pas se faire car la dernière instruction SEP R (α) renvoie non pas au programme SP_β mais au programme principal. Par contre, on peut imaginer qu'un sous-programme quelconque appelle par cette méthode d'autres sous-programmes *qui lui sont propres*. Tout sous-programme doit « connaître » le compteur du programme appelant afin d'effectuer le retour.

Il n'en reste pas moins que *cette technique constitue la méthode à choisir* pour des programmes peu ambitieux.

III. La technique standard d'appel et de retour

III. 1. Caractères généraux de la méthode

Cette technique standard d'appel et de retour, plus avancée, offre certains avantages sur la précédente :

— les sous-programmes peuvent être en nombre quelconque : pas de limitation par le nombre des registres disponibles car c'est le même registre interne qui sert de compteur pour le programme principal et pour tous les sous-programmes standards ;

— les sous-programmes peuvent être imbriqués à l'infini : un sous-programme peut appeler n'importe quel autre sous-programme ;

— le programme appelant *peut passer des données* au sous-programme.

Ceci constitue une originalité extrêmement intéressante de cette technique ; nous en donnerons au moins un exemple d'exploitation (sous-programme de temporisation du moniteur).

La méthode n'est cependant pas sans inconvénients :

— elle réserve trois registres pour les enchaînements (appel et retour) ;

— elle nécessite un temps d'exécution relativement long pour ces enchaînements.

III. 2. Affectations des registres

a) *Pointeur de pile : registre R (2)*

La technique standard d'appel et de retour utilise un espace de mémoire vive appelé pile pour sauvegarder des adresses.

Rappelons ici qu'une pile consiste en un certain nombre de lignes de RAM, pointées par un registre R spécifié par l'utilisateur en fixant le contenu du registre X (le registre R (X) choisi pour cette fonction s'appelle pointeur de pile ou stack pointer en anglais). Cette pile est gérée suivant la technique « dernier entré, premier sorti » (LIFO : last in, first out).

Le pointeur contient une adresse qui permet au microprocesseur de trouver le « sommet » de la pile. Quand un octet est stocké dans la pile, il l'est à l'adresse indiquée par ce pointeur ; celui-ci est alors immédiatement décrémenté pour désigner la ligne RAM libre juste au-dessus de la pile de manière à préparer le stockage d'un nouvel octet (ceci justifie l'existence de l'instruction Store via X and Decrement). Réciproquement, pour extraire l'information de la pile, le pointeur est d'abord incrémenté, l'octet pointé est lu et chargé dans un registre. La ligne pointée (sommet courant de la pile) est alors libre pour recevoir un autre octet. Bien entendu, une suite d'octets peut être lue dans la pile par une succession d'incrémentations du pointeur et de transferts dans D via X (l'instruction Load via X and advance facilite l'opération).

Le choix du registre R (2) comme pointeur de pile est le plus naturel étant donnée la manière dont les interruptions sont traitées par le 1802 — le programmeur est malgré tout libre de son choix.

La zone mémoire réservée à la pile doit être suffisamment grande pour contenir deux octets pour chaque niveau d'imbrication en plus des octets que le programme lui-même sauvegarde en pile (si par exemple, le programme principal peut appeler un sous-programme, qui lui-même peut en appeler un deuxième, celui-ci pouvant en appeler un autre, le niveau d'imbrication est 3).

b) *Compteur de programme de base : registre R (3)*

Le registre R (3) est choisi, arbitrairement, comme compteur de *tout* programme principal et de *tous* les sous-programmes standard.

Attention : le programme moniteur lance les programmes utilisateurs avec R (3) comme compteur de programme. Il est indispensable que vous vous interdisiez toute autre utilisation de ce registre si vous vous aidez du moniteur.

c) *Compteurs de programmes spécialisés : registres R (4) et R (5)*

La technique standard d'appel et de retour utilise deux sous-programmes d'enchaînement, l'un pour l'appel, l'autre pour le retour, gérés suivant la méthode du paragraphe II.

Le registre R (4) est choisi comme compteur du programme du SP d'appel, alors que R (5) est attribué au SP de retour.

R (4) et R (5) se doivent d'être initialisés par le programme principal avant l'appel d'un SP.

d) *Registre de sauvegarde interne : registre R 6)*

III. 3. Appel d'un sous-programme

Du programme appelant, l'appel se fait grâce à l'utilisation de l'instruction SEP R (4). Dans le programme appelant, le code D 4 de cette instruction inscrit à l'adresse Z est suivi des deux octets W_1 et W_0 de l'adresse W du sous-programme standard à appeler. A la suite — on dit aussi en ligne — on trouve des données qui sont à faire passer du programme appelant au sous-programme (indications à fournir au sous-programme) : voir figure VII. 3.

L'instruction SEP R (4) renvoie au programme spécifique d'appel en laissant le compteur R (3) du programme principal avec le contenu Z+1 (organigramme sur la figure VII. 4). Dans ce sous-programme d'appel, le registre R (2) est d'abord désigné comme pointeur de pile (le programme appelant a pu modifier le contenu de X). *Puis le contenu courant de R (6) est stocké dans la pile* (l'intérêt de cette opération ne peut être compris qu'a posteriori). Le contenu de R (3) (c'est-à-dire Z+1) est recopié dans R (6) : il s'agit d'une mesure de sauvegarde. Enfin les deux octets W_1 et W_0 de l'adresse W du sous-programme standard à appeler sont transférés dans R (3). Notez que l'incrémentation finale de R (6) laisse celui-ci avec le contenu Z+3.

L'instruction SEP R (3) renvoie au sous-programme. L'incrémentation de R (4) pendant le cycle de recherche a réinitialisé automatiquement celui-ci à l'adresse de départ du SP d'appel.

Si le sous-programme appelé a besoin de données écrites dans le programme appelant juste après l'adresse du sous-programme, il ira les chercher au moyen d'instructions LOAD ADVANCE via R (6) : les données

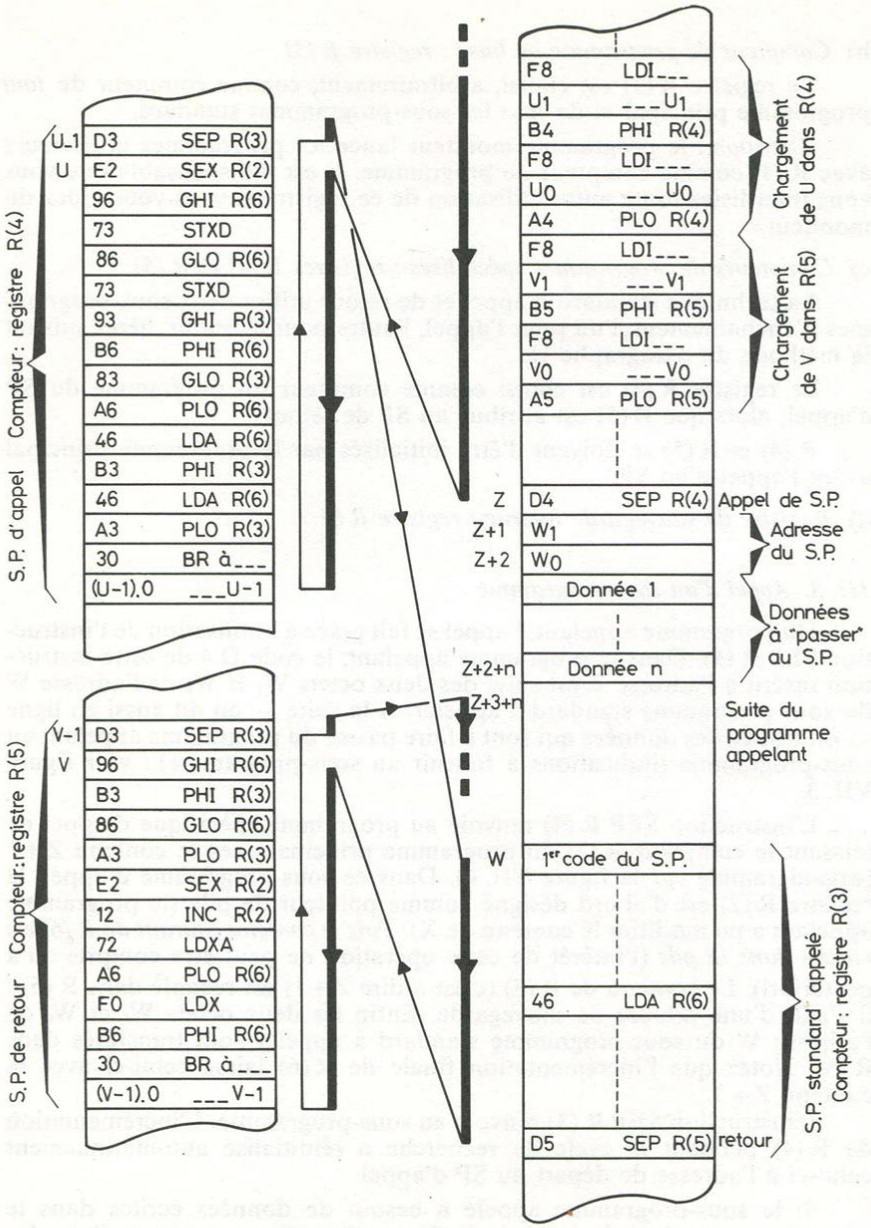


Fig. VII. 3. Appel d'un sous-programme et retour au programme appelant par la technique standard

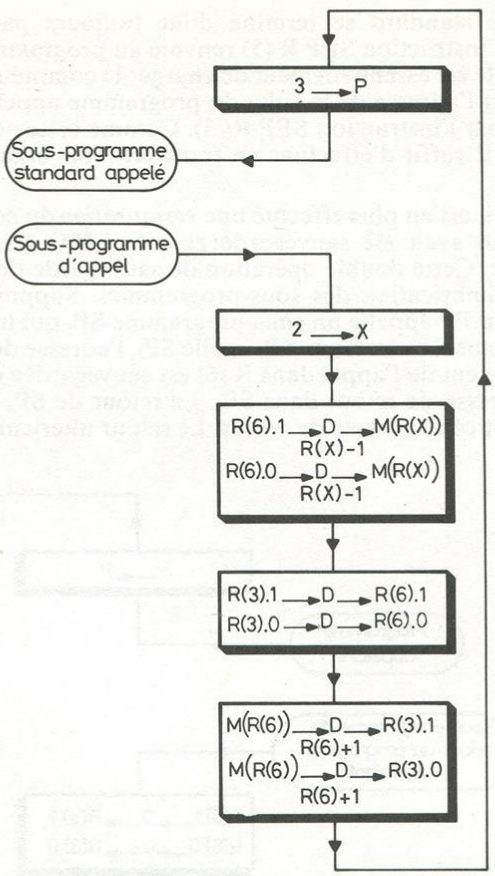


Fig. VII. 4. Organigramme du sous-programme spécifique d'appel d'un sous-programme standard

successives à transmettre, pointées par le registre R (6) sont ainsi transférées dans l'accumulateur D afin d'être utilisées par le sous-programme, et le registre R (6) est incrémenté pour préparer un nouveau transfert. Si n données sont transmises au sous-programme standard, en sortant de ce sous-programme, le registre R (6) contient Z + n + 3, adresse de l'instruction suivante du programme appelant.

III. 4. Retour d'un sous-programme standard au programme appelant

Quand le sous-programme standard a effectué sa fonction propre, le retour au programme appelant se fait en utilisant l'instruction SEP R (5) (un

sous-programme standard se termine donc toujours par le code D 5). L'exécution de l'instruction SEP R (5) renvoie au programme spécifique de retour dont le rôle est essentiellement de charger le compteur de programme de base R (3) par l'adresse de la suite du programme appelant avant de lui rendre la main par l'instruction SEP R(3). Comme cette adresse est contenue dans R (6), il suffit d'effectuer un transfert (voir organigramme de la figure VII. 5).

On note qu'il est en plus effectué une *restauration* du contenu initial du registre R (6) qui avait été sauvegardé en pile LIFO lors de l'appel du sous-programme. Cette double opération de sauvegarde de R (6) et restauration permet l'imbrication des sous-programmes. Supposons qu'un programme principal PP appelle un sous-programme SP_1 qui lui-même appelle un sous-programme SP_2. Lorsque SP_1 apelle SP_2, l'adresse de retour dans PP contenue au moment de l'appel dans R (6) est sauvegardée dans la pile puis R (6) reçoit l'adresse de retour dans SP_1. Le retour de SP_2 dans SP_1 se fait car R (6) contient cette adresse de retour. Le retour ultérieur de SP_1 dans PP

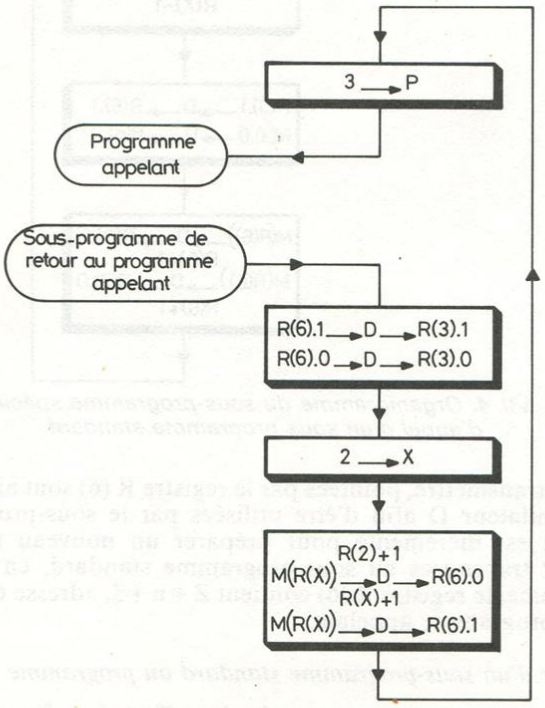


Fig. VII. 5. Organigramme du sous-programme spécifique de retour au programme appelant

ne peut s'effectuer que grâce à la restitution à R (6) de l'adresse de retour stockée en mémoire. Bien entendu, cet exemple se généralise à un niveau quelconque d'imbrication de sous-programmes.

III. 5. Remarques à propos du moniteur

a) Le programme moniteur utilise cette technique standard d'appel et de retour. Il contient donc les deux routines d'appel et de retour écrites respectivement à partir des adresses 001A et 002A ainsi que les instructions nécessaires à l'initialisation des compteurs de programme spécifiques R (4) et R (5).

Dans le cadre de la mise au point d'un programme, l'utilisateur fait appel au moniteur et il n'a donc pas à réécrire ces routines. L'exemple pratique fourni au paragraphe IV du chapitre 5 illustre cette facilité offerte à l'utilisateur.

b) Un des sous-programmes du moniteur (le sous-programme de temporisation, écrit à partir de l'adresse 0100) se sert de données qui sont transmises à partir du programme appelant. L'appel du sous-programme de temporisation se fait pratiquement en inscrivant en mémoire le code D 4 de SEP R (4) suivi des deux octets 01 et 00 de l'adresse de ce sous-programme *puis des deux octets à charger dans le compteur de boucle* (données nécessaires au réglage de la durée de temporisation). Le transfert de ces données est assuré par les deux instructions LDA R (6) écrites dans le sous-programme de temporisation.

c) Avec ces explications, le lecteur est à même de comprendre entièrement l'exemple du paragraphe IV du chapitre 5. Nous l'invitons à faire cet effort de compréhension afin de pouvoir utiliser ces techniques pour ses programmes propres.

d) Dans tous les organigrammes des prochains chapitres, les sous-programmes sont repérés par un cadre avec des barres verticales doublées.

e) *Attention :* les durées de temporisation du programme moniteur ne sont correctes *que si l'on utilise une horloge munie d'un quartz 2 MHz.*

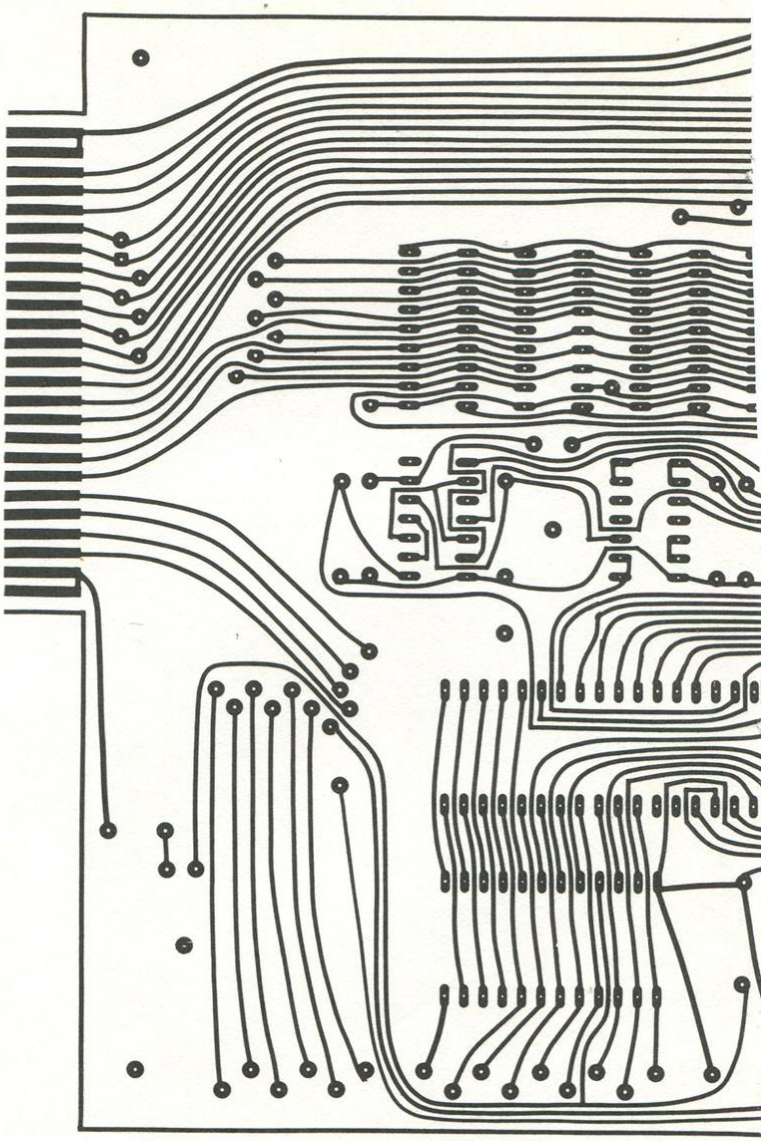
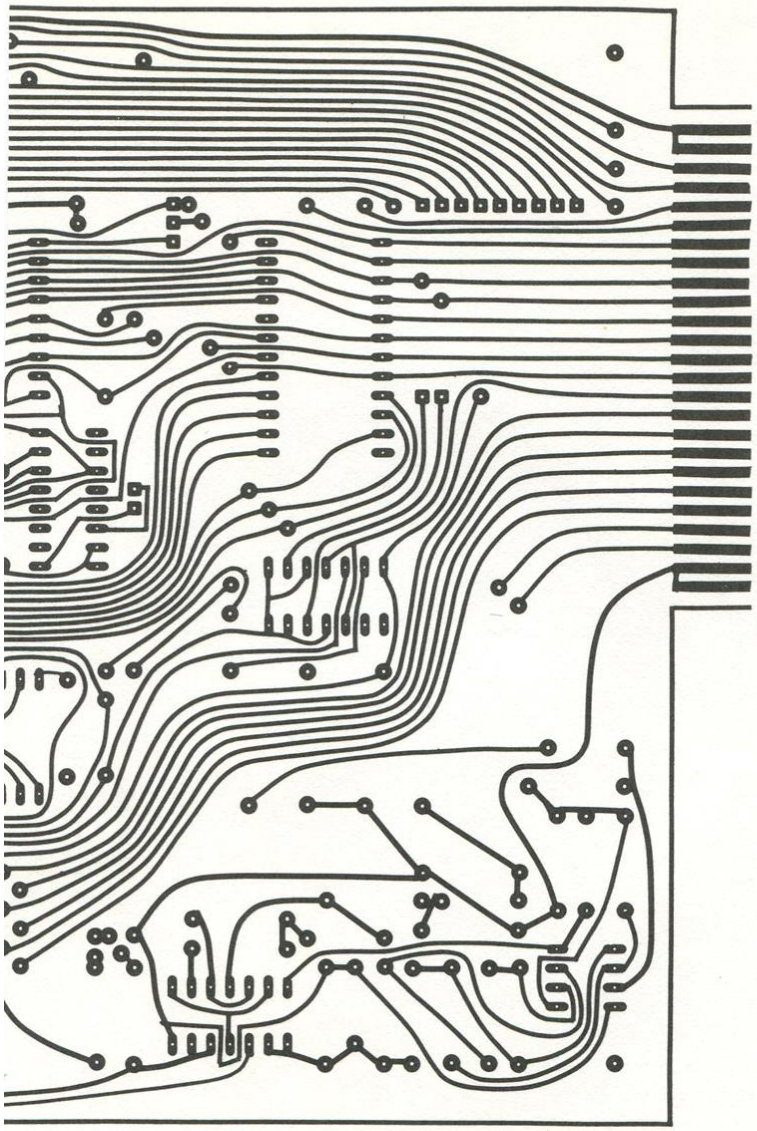


Fig. III. 9. Circuit imprimé face dessous (cuivre)



(Voir pages 80-81 côté composants).

Chapitre 8

LE MONITEUR : DESCRIPTION LOGICIELLE

Le chapitre 5 a permis de mettre en évidence la fonction du moniteur et en a fourni la procédure d'utilisation. En utilisant les éléments fondamentaux développés dans les chapitres 6 et 7, nous comptons expliquer ici comment est construit le programme moniteur. En dehors de l'intérêt bien légitime que le lecteur portera à la compréhension du système qu'il a éventuellement construit, celui-ci y verra un exemple, déjà conséquent, de programme. Il y puisera sans aucun doute des idées et des méthodes pour le développement de ses propres programmes d'application.

I. Description générale

I. 1. Utilisation de la mémoire

Le programme moniteur proprement dit est écrit dans une EPROM 2716 et réside presque entièrement entre les adresses 0000 et 01FF.

Quelques lignes de la RAM 128 octets résidant aux adresses 0C00 à 0C7F sont requises par le moniteur lui-même pour stocker temporairement des données (fig. VIII. 1). Bien entendu, lors de l'étude et de la mise au point d'un programme utilisateur à l'aide du moniteur, il faut s'interdire l'utilisation de ces lignes RAM. Notez que le moniteur proprement dit n'entraîne jamais plus de deux niveaux de sous-programmes (le programme principal utilise des sous-programmes qui eux-mêmes appellent éventuellement des sous-programmes, ces derniers n'en appelant pas d'autres). Dans ces conditions, la pile LIFO n'occupe jamais plus de quatre lignes.

Des développements du moniteur, que nous évoquerons plus loin, occupent les adresses 0200 à 03FF de l'EPROM 2716 et utilisent les six lignes 0C00 à 0C05 de la RAM 126 octets (fig. VIII. 1).

I. 2. Utilisation des registres internes du microprocesseur

Nous avons essayé de faire l'utilisation la plus large possible des seize registres R (W) internes du microprocesseur en tant que pointeurs de zones particulières de la mémoire, compteurs de boucle, registres de sauvegarde

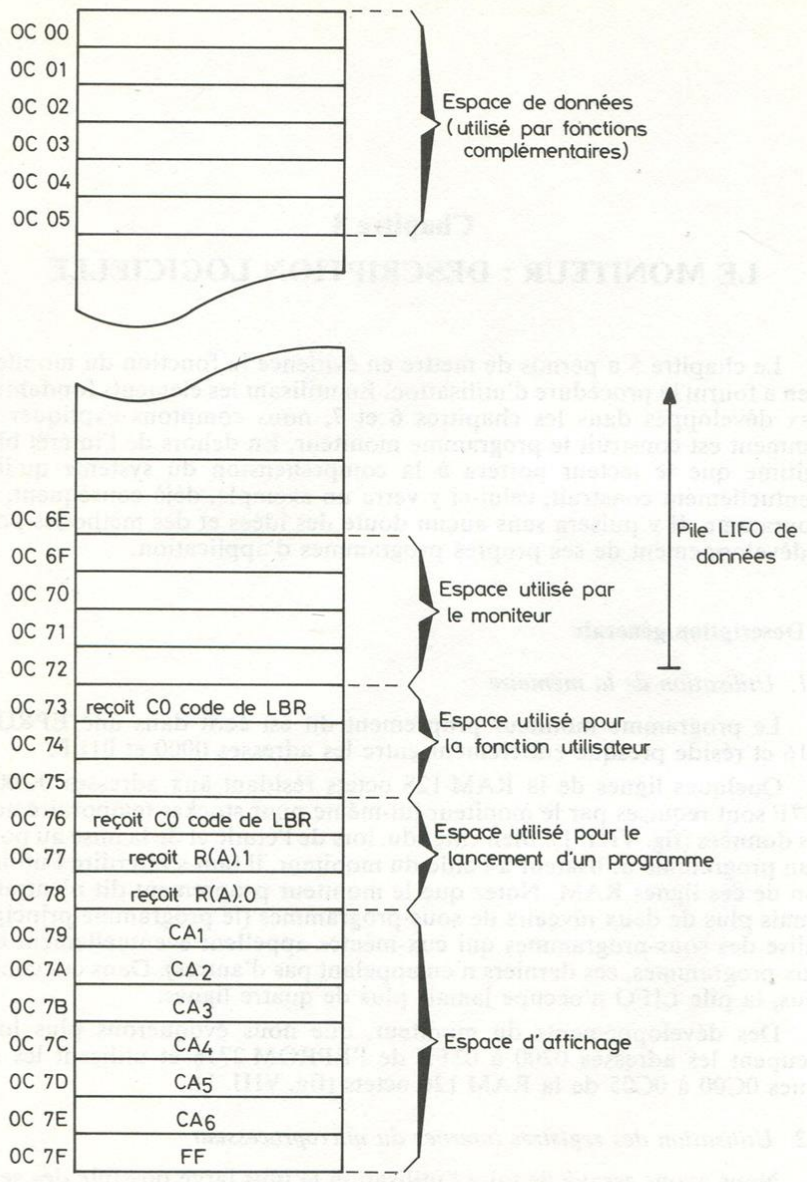


Fig. VIII. 1. Utilisation de la RAM 128 octets

ou registres de travail. Au cours de l'étude du moniteur, on verra que les affectations sont les suivantes :

R (0) Compteur de programme après mise en service générale ou remise à zéro.
R (1) Non utilisé par le moniteur proprement dit.
R (2) Pointeur de la pile LIFO.
R (3) Compteur de programme pour le programme principal et les sous-programmes ordinaires.
R (4) Compteur de programme pour le sous-programme particulier d'appel à un sous-programme ordinaire.
R (5) Compteur de programme pour le sous-programme particulier de retour d'un sous-programme ordinaire vers le programme appelant.
R (6) Registre de sauvegarde utilisé dans la technique de gestion des sous-programmes.
R (7) Pointeur du circuit 8255.
R (8) Pointeur de l'espace des codes des touches de caractères hexadécimaux et des codes d'affichages en ROM moniteur.
R (9) Compteur de boucle d'usage général. R (9).1 est utilisé pour des sauvegardes.
R (A) Registre d'adresse du moniteur (contient l'adresse affichée).
R (B) Non utilisé par le moniteur proprement dit.
R (C) Compteur de boucle du sous-programme de temporisation.
R (D) Registre de donnée du moniteur (R (D).0 reçoit la donnée affichée).
R (E) R (E).0 est utilisé pour la formation du code d'une touche enfoncée R (E).1 contient l'indicateur d'utilisation de la touche M (voir ci-dessous).
R (F) Pointeur des espaces d'affichage et de branchement de la RAM 128 octets.

On se souvient qu'il a été attribué deux fonctions différentes à la touche P :

— si l'on a, après initialisation, utilisé la touche M, une pression sur la touche P provoque l'incrémentation du registre d'adresse R (A) et l'affichage de la nouvelle adresse et du contenu correspondant de la mémoire ;

— si, après initialisation, on n'a jamais utilisé la touche M, une pression sur la touche P entraîne le lancement du programme à l'adresse contenue dans le registre d'adresse R (A).

Pour que le microprocesseur — à qui sont confiées ces diverses tâches — sache dans quel sens orienter son action à la suite d'une pression de la touche P, il faut que l'utilisation de la touche M entraîne le positionnement d'un indicateur. Pour des raisons de commodité de programmation, on a décidé que c'est le bit de poids fort du demi-registre R (E).1 qui jouera le rôle d'indicateur d'utilisation de la touche M. On notera IM cet indicateur. Une initialisation du moniteur met automatiquement IM à 0 ; quand on

appuie sur la touche M, IM se met automatiquement à 1 après relâchement. Une pression sur P entraîne toujours une consultation de IM.

I. 3. Organisation générale du moniteur

Elle est décrite par l'organigramme de la figure VIII. 2.

Après mise en service générale du système ou action sur le bouton-poussoir de remise à zéro, le moniteur s'engage dans une séquence d'initialisations dans laquelle les registres utilisés par le moniteur sont initialisés, le port C du PPI 8255 est programmé en sortie, l'espace d'affichage en RAM 128 octets est chargé par les codes nécessaires à l'affichage du mot PrEt et l'indicateur IM d'utilisation de la touche M est mis à 0.

L'affichage (initialement du mot PrEt), la scrutation et l'encodage du clavier se font suivant les techniques décrites au chapitre 6.

La sortie du bloc affichage-scrutation-encodage ne s'effectue que si une touche a été enfoncée puis relâchée. Le moniteur détermine la nature de cette touche (touche fonction ou touche de caractère hexadécimal) à partir de son code qui a été construit dans le registre R (E).0.

S'il s'agit d'une touche de fonction, le décodage aboutit à l'identification de cette fonction et l'on effectue un branchement au programme d'exécution de cette fonction.

S'il s'agit d'une touche de caractère hexadécimal, le moniteur en effectue le décodage (c'est-à-dire qu'il identifie le caractère hexadécimal de la touche) et il forme le code d'affichage de ce caractère. L'indicateur d'utilisation de la touche M est alors consulté :

— si IM = 1, c'est que l'on a préalablement utilisé la touche M et que la pression d'une touche de caractère a pour but de modifier un octet de RAM. Le moniteur réalise alors la modification du registre de donnée R (D).0 (le quartet initial de faible poids remplace le quartet initial de fort poids ; le quartet correspondant au caractère hexadécimal remplace le quartet initial de faible poids), de l'espace d'affichage de la donnée (CA_6 initial remplace CA_5 initial et le code d'affichage du caractère de la touche pressée remplace CA_6 initial), et bien entendu de la mémoire à l'adresse affichée (la donnée modifiée contenue dans R (D).0 est transférée dans la mémoire à l'adresse indiquée par le registre R (A)) ;

— si IM = 0, la pression d'une touche doit modifier une adresse, contenue dans le registre R (A) et affichée sur les quatre afficheurs de gauche. Le moniteur réalise donc la modification de R (A) (décalage à gauche de quatre bits, le quartet de fort poids étant perdu, et remplacement du quartet de faible poids par le quartet correspondant au caractère de la touche actionnée), et la modification de l'espace d'affichage de l'adresse (décalage $CA_1 \leftarrow CA_2 \leftarrow CA_3 \leftarrow CA_4$, remplacement de CA_4 initial par le code d'affichage de la touche pressée).

Quand l'une où l'autre de ces fonctions est réalisée, on revient à l'affichage et à une nouvelle scrutation du clavier.

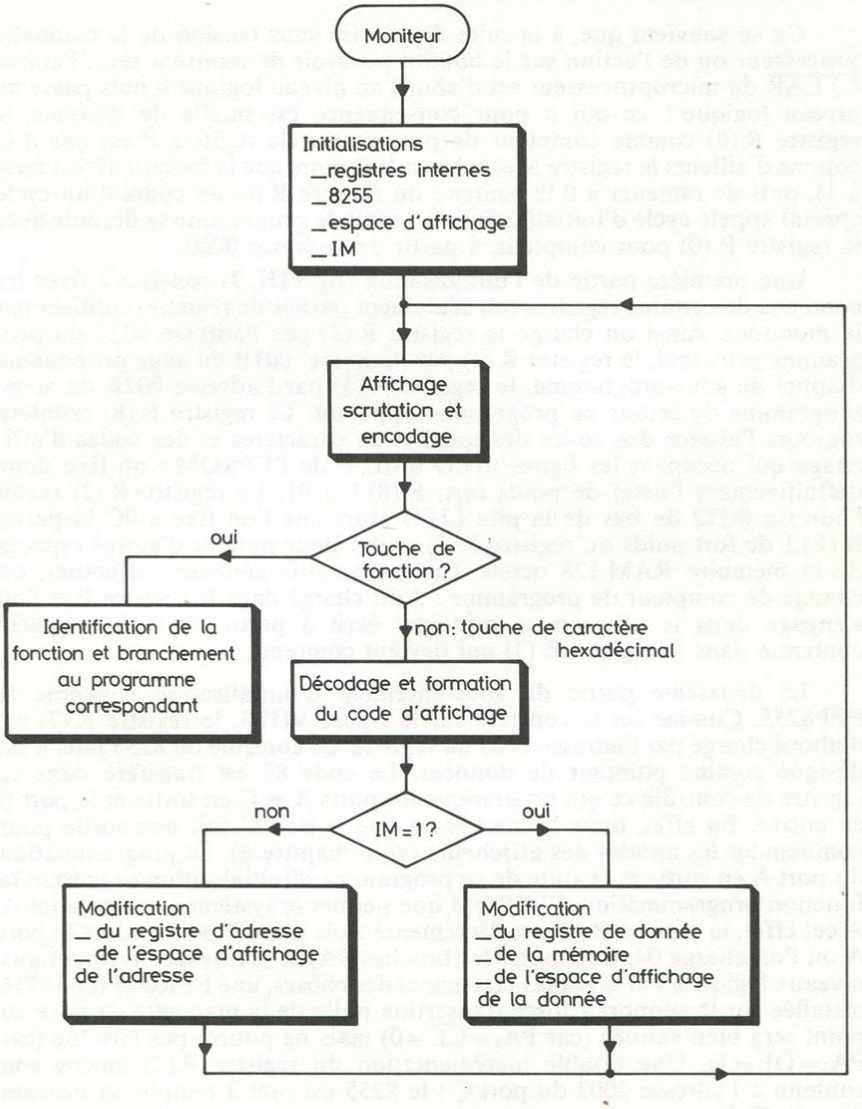


Fig. VIII. 2. Organisation générale du moniteur

II. Description des différentes parties du moniteur

II. 1. Sous-ensemble Initialisations

On se souvient que, à la suite de la mise sous tension de la maquette processeur ou de l'action sur le bouton-poussoir de remise à zéro, l'entrée $\overline{\text{CLEAR}}$ du microprocesseur est d'abord au niveau logique 0 puis passe au niveau logique 1 ce qui a pour conséquence essentielle de désigner le registre R (0) comme compteur de programme (le registre P est mis à 0, comme d'ailleurs le registre X et la bascule Q alors que la bascule IE est mise à 1), puis de ramener à 0 le contenu du registre R (0) au cours d'un cycle spécial appelé cycle d'initialisation. Cela fait, le programme se déroule avec le registre R (0) pour compteur, à partir de l'adresse 0000.

Une première partie de l'initialisation (fig.VIII. 3) consiste à fixer les contenus de certains registres (ou seulement parties de registres) utilisés par le moniteur. Ainsi on charge le registre R (3) par l'adresse 0037 du programme principal, le registre R (4) par l'adresse 001B du sous-programme d'appel de sous-programme, le registre R (5) par l'adresse 002B du sous-programme de retour au programme appelant. Le registre R (8) pointera toujours l'espace des codes des touches de caractères et des codes d'affichage qui occupent les lignes 01EO à 01FF de l'EPROM : on fixe donc définitivement l'octet de poids fort, R (8).1 à 01. Le registre R (2) reçoit l'adresse 0C72 du bas de la pile LIFO alors que l'on fixe à 0C la partie R (F).1 de fort poids du registre R(F) utilisé pour pointer d'autres espaces de la mémoire RAM 128 octets. Ces préparatifs généraux effectués, on change de compteur de programme : 3 est chargé dans le registre P et l'on s'engage dans le programme principal écrit à partir de l'adresse 0037 contenue dans le registre R (3) qui devient compteur.

La deuxième partie du sous-ensemble d'initialisation concerne le PPI 8255. Comme on le constate sur la figure VIII.3, le registre R (7) est d'abord chargé par l'adresse 2003 du registre de contrôle du 8255 puis il est désigné comme pointeur de données. Le code 82 est transféré dans ce registre de contrôle ce qui programme les ports A et C en sortie et le port B en entrée. En effet, nous avons besoin que le port C soit une sortie pour commander les anodes des afficheurs (voir chapitre 6). La programmation du port A en sortie et la suite de ce programme d'initialisation préparent la fonction programmation d'EPROM que permet le système (voir plus loin). A cet effet, le registre R (7) est décrémenté trois fois ; il pointe alors le port A où l'on charge 04. Ceci porte les broches PA_2 et PA_5 respectivement aux niveaux logiques 1 et 0. Dans l'état actuel des choses, une EPROM type 2716 installée sur le support à force d'insertion nulle de la maquette de mise au point sera bien validée (car $PA_5 = \overline{CE} = 0$) mais ne pourra pas être lue (car $PA_2 = \overline{OE} = 1$). Une double incrémentation du registre R (7) amène son contenu à l'adresse 2002 du port C : le 8255 est prêt à remplir sa mission dans l'affichage.

La troisième partie remplit l'espace d'affichage par les codes d'affi-

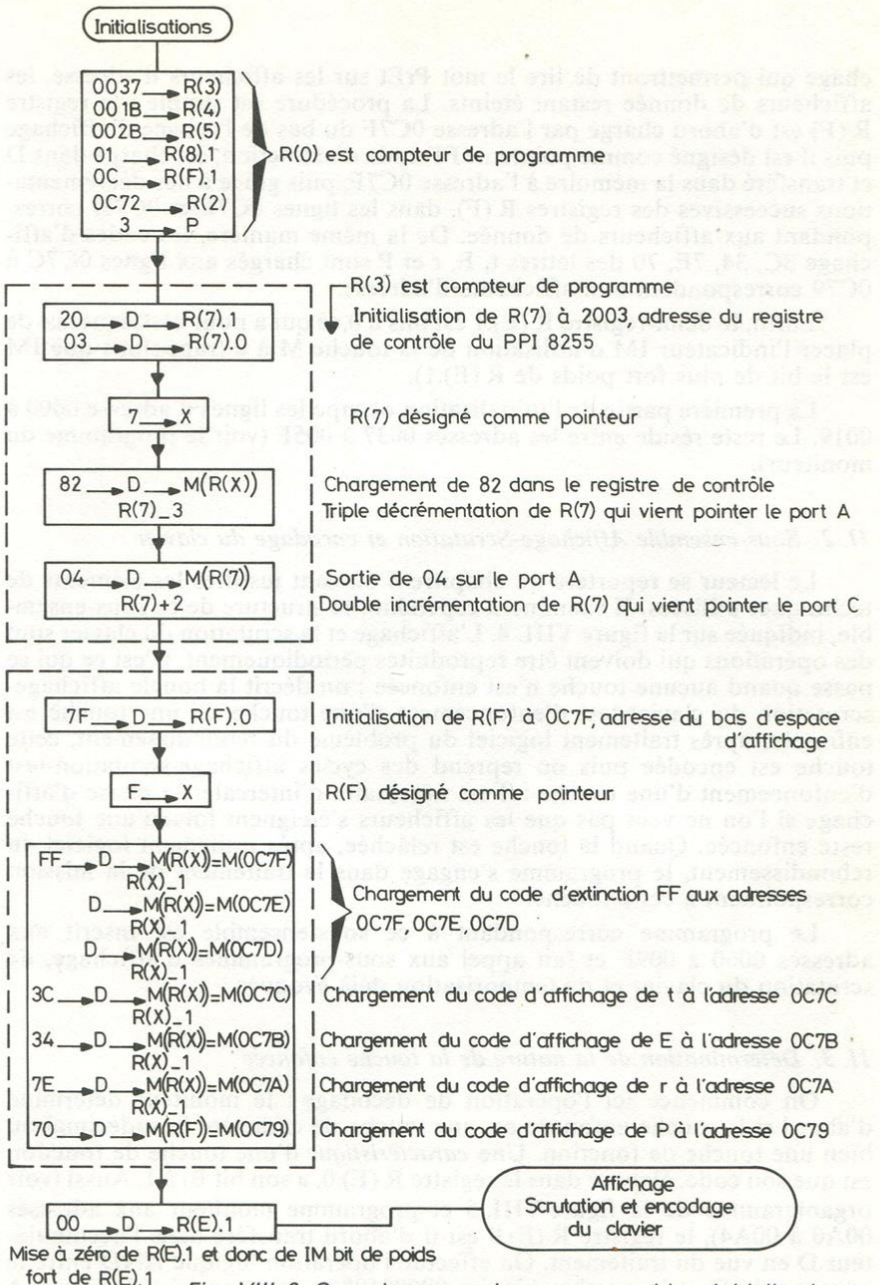


Fig. VIII. 3. Organigramme du sous-ensemble « initialisations »

chage qui permettront de lire le mot PrEt sur les afficheurs d'adresse, les afficheurs de donnée restant éteints. La procédure est simple : le registre R (F) est d'abord chargé par l'adresse 0C7F du bas de l'espace d'affichage puis il est désigné comme pointeur. FF code d'extinction, est chargé dans D et transféré dans la mémoire à l'adresse 0C7F, puis grâce à des décrémentations successives des registres R (F), dans les lignes 0C7E et 0C7D correspondant aux afficheurs de donnée. De la même manière, les codes d'affichage 3C, 34, 7E, 70 des lettres t, E, r et P sont chargés aux lignes 0C7C à 0C79 correspondant aux afficheurs d'adresse.

Enfin, le demi-registre R (E).1 est mis à 0,ce qui a pour conséquence de placer l'indicateur IM d'utilisation de la touche M à 0 (rappelons que IM est le bit de plus fort poids de R (E).1).

La première partie de l'initialisation occupe les lignes d'adresse 0000 à 0019. Le reste réside entre les adresses 0037 à 005F (voir le programme du moniteur).

II. 2. Sous-ensemble Affichage-Scrutation et encodage du clavier

Le lecteur se reportera au chapitre 6 où sont justifiés les éléments de techniques utilisées. Il examinera cependant la structure de ce sous-ensemble, indiquée sur la figure VIII. 4. L'affichage et la scrutation du clavier sont des opérations qui doivent être reproduites périodiquement. C'est ce qui se passe quand aucune touche n'est enfoncée : on décrit la boucle affichage-scrutation du clavier-test d'enfoncement d'une touche. Si une touche est enfoncée, après traitement logiciel du problème du rebondissement, cette touche est encodée puis on reprend des cycles affichage-scrutation-test d'enfoncement d'une touche : il est nécessaire d'intercaler la phase d'affichage si l'on ne veut pas que les afficheurs s'éteignent lorsqu'une touche reste enfoncée. Quand la touche est relâchée, après traitement logiciel du rebondissement, le programme s'engage dans le traitement de la mission correspondant à cette touche.

Le programme correspondant à ce sous-ensemble est inscrit aux adresses 0060 à 009F et fait appel aux sous-programmes d'affichage, de scrutation du clavier et de temporisation déjà évoqués.

II. 3. Détermination de la nature de la touche enfoncée

On commence ici l'opération de décodage : le moniteur détermine d'abord si la touche enfoncée est une touche de caractère hexadécimal ou bien une touche de fonction. Une *caractéristique* d'une touche de fonction est que son code, élaboré dans le registre R (E).0, a son bit B_3 à 1. Aussi (voir organigramme de la figure VIII. 5 et programme moniteur aux adresses 00A0 à 00A4), le registre R (E).0 est-il d'abord transféré dans l'accumulateur D en vue du traitement. On effectue l'opération logique AND entre le contenu de D et le nombre binaire 00000100 (soit 04 en héxadécimal). A

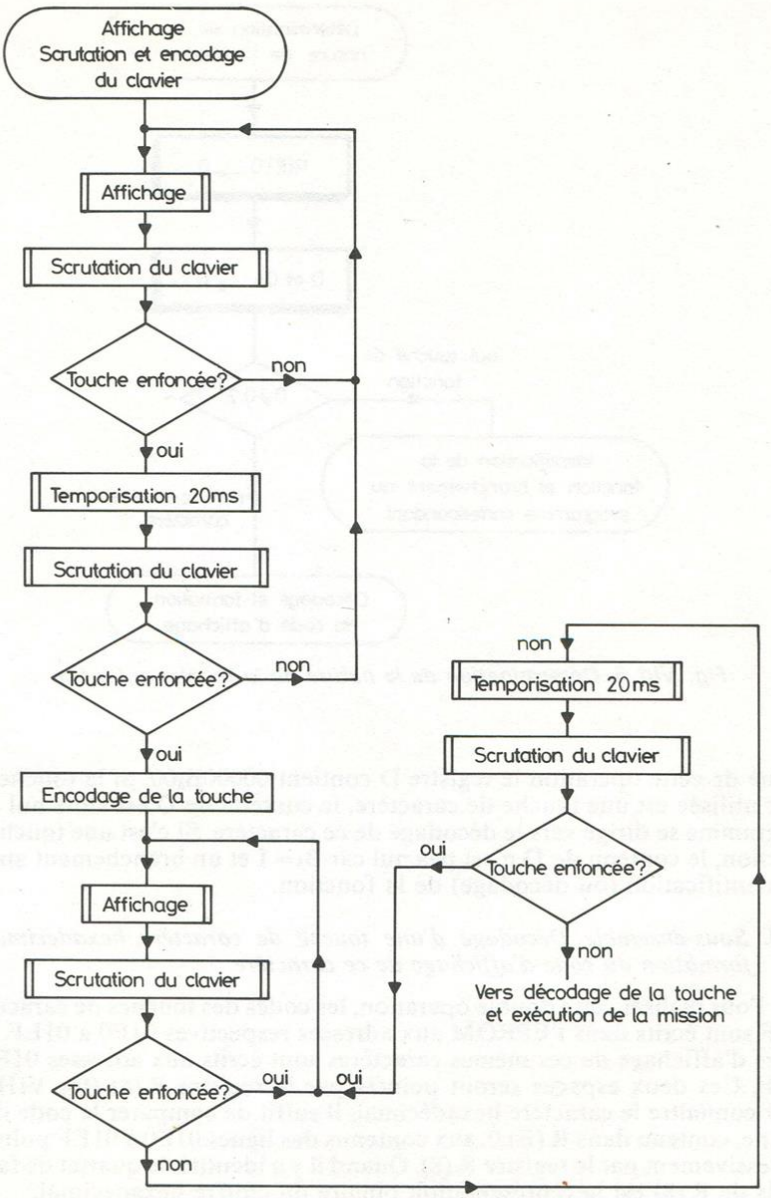


Fig. VIII. 4. Organigramme du sous-ensemble « affichage scrutation-encodage »

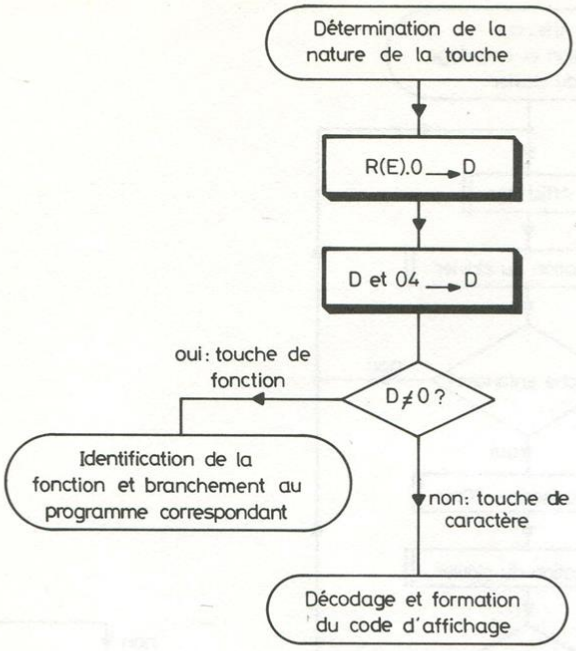


Fig. VIII. 5. Détermination de la nature de la touche enfoncée

l'issue de cette opération le registre D contient $00000B_300$. Si la touche qui a été utilisée est une touche de caractère, le contenu de D est alors nul et le programme se dirige vers le décodage de ce caractère. Si c'est une touche de fonction, le contenu de D n'est pas nul car $B_3 = 1$ et un branchement amène à l'identification (ou décodage) de la fonction.

II. 4. Sous-ensemble Décodage d'une touche de caractère hexadécimal et formation du code d'affichage de ce caractère

Pour réaliser cette double opération, les codes des touches de caractères 0 à F sont écrits dans l'EPROM aux adresses respectives 01E0 à 01EF. Les codes d'affichage de ces mêmes caractères sont écrits aux adresses 01F0 à 01FF. Ces deux espaces seront pointés par le registre R (8) (fig. VIII. 6). Pour connaître le caractère hexadécimal, il suffit de comparer le code de la touche, contenu dans R (E).0, aux contenus des lignes 01E0 à 01EF pointées successivement par le registre R (8). Quand il y a identité, le quartet de faible poids de R (8) est la représentation binaire du chiffre hexadécimal.

Le code d'affichage de ce chiffre peut être lu dans la mémoire 16 lignes

Pointeur :

Registre R (8)

01E0	Code de la touche	0	10
E1	«	1	11
E2	«	2	12
E3	«	3	13
E4	«	4	20
E5	«	5	21
E6	«	6	22
E7	«	7	23
E8	«	8	40
E9	«	9	41
EA	«	A	42
EB	«	B	43
EC	«	C	80
ED	«	D	81
EE	«	E	82
EF	«	F	83
01F0	Code d'affichage de	0	21
F1	«	1	EB
F2	«	2	32
F3	«	3	A2
F4	«	4	E8
F5	«	5	A4
F6	«	6	24
F7	«	7	E3
F8	«	8	20
F9	«	9	E0
FA	«	A	60
FB	«	B	2C
FC	«	C	35
FD	«	D	2A
FE	«	E	34
FF	«	F	74

Fig. VIII. 6. Espaces des codes des touches de caractères et des codes d'affichage de ces caractères

plus loin en ajoutant 10 (en hexadécimal) au contenu courant du registre pointeur R (8) (ce qui, notons-le, ne modifie pas le quartet de faible poids). On a décidé de transférer ce code d'affichage dans le registre R(E).0 où il vient remplacer le code de la touche.

On aboutit ainsi à l'organigramme de la ligne VIII. 7. R (8) est d'abord initialisé à 01E0 et le registre X est mis à 8 (nécessité pour l'opération de comparaison qui utilise la soustraction). Le code de la touche contenu dans R (E).0 est comparé à l'octet mémoire pointé par R (8). S'il n'y a pas identité, le registre R (8) est incrémenté et l'on reprend la comparaison. S'il y a

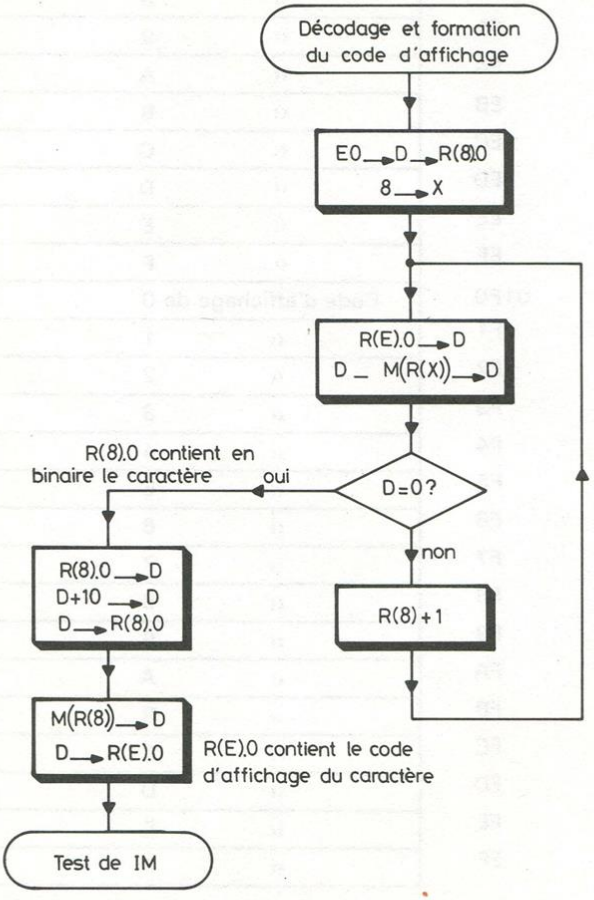


Fig. VIII. 7. Organigramme de décodage et de formation du code d'affichage d'un caractère hexadécimal

identité, R (8) est augmenté de 10 (en base 16) et vient pointer le code d'affichage qui est transféré dans R (E).0.

Le programme correspondant est écrit aux adresses 00A5 à 00B5.

II. 5. Test de l'indicateur d'utilisation de la touche M

L'organigramme de la figure VIII. 8 auquel correspondent les lignes 00B6 à 00B9 du programme moniteur montre que l'indicateur IM (bit B7 de R (E).1) est placé dans DF grâce à un transfert de R (E).1 dans D puis un décalage à gauche de D. C'est alors DF qui est testé : si DF = 0, soit IM = 0, la séquence se poursuit par le programme de modification d'adresse ; si DF = 1, soit IM = 1, un branchement s'effectue au programme de modification de donnée.

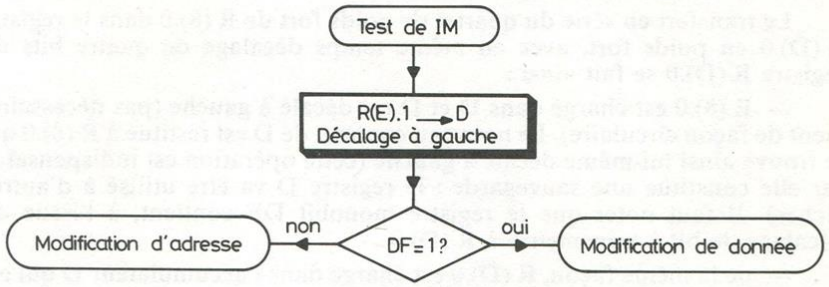


Fig. VIII. 8. Organigramme du test de l'indicateur IM

II.6. Sous-ensemble Modification de donnée

Comme l'indique l'organigramme de la figure VIII. 9, ce sous-ensemble remplit quatre fonctions :

a) *Décalage de l'espace d'affichage de la donnée.*

Comme l'on désire décaler l'affichage de la donnée à gauche et afficher sur l'afficheur 6 le caractère de la touche enfoncée, il faut d'abord écrire le code d'affichage initial CA_6 à la place de CA_5. Cette mission est remplie par le sous-programme de décalage $CA_5 \leftarrow CA_6$. Le registre R (F) est d'abord initialisé à 0C7E, adresse où se trouve écrit le code d'affichage CA_6. Ce code est inscrit à la nouvelle adresse 0C7D.

b) *Stockage du code d'affichage du caractère de la touche enfoncée.*

On a vu que ce code a été formé dans le demi-registre R (E).0. Le registre pointeur R (F) est incrémenté pour qu'il contienne à nouveau l'adresse 0C7E à laquelle le code d'affichage doit être écrit. Puis le code est transféré du registre R (E).0 dans la mémoire, via l'accumulateur D.

c) *Mise à jour du registre de donnée R (D).0*

Il faut que l'octet contenu dans le registre R (D).0 soit conforme au nombre qui sera affiché en hexadécimal lors du prochain affichage. Il est nécessaire là encore de procéder à un décalage d'un quartet (quatre bits) du registre R (E).0 puis d'y écrire en faible poids le quartet correspondant à la touche enfoncée. Ceci est effectué grâce à un sous-programme spécifique (fig. VIII. 9). Le registre R (9).0 y est initialisé à quatre pour compter quatre boucles. On sait que le quartet correspondant à la touche de caractère hexadécimal enfoncée est le quartet de poids faible de R (8).0. Pour le transférer dans R (D).0, on utilise le mode série. A cet effet, on commence par décaler le registre R (8).0 de manière que le quartet à transférer occupe la position de poids fort (bien entendu, comme il n'existe pas d'instruction de décalage de registre, il faut d'abord transférer dans D puis ramener dans R (8).0 après quatre décalages à gauche.)

Le transfert en série du quartet de poids fort de R (8).0 dans le registre R (D).0 en poids fort, avec en même temps décalage de quatre bits du registre R (D).0 se fait ainsi :

— R (8).0 est chargé dans D et D est décalé à gauche (pas nécessairement de façon circulaire). Le nouveau contenu de D est restitué à R (8).0 qui se trouve ainsi lui-même décalé à gauche (cette opération est indispensable car elle constitue une sauvegarde : le registre D va être utilisé à d'autres tâches). Il faut noter que le registre monobit DF contient, à l'issue du décalage, le bit à transmettre à R (D).0 ;

— de la même façon, R (D).0 est chargé dans l'accumulateur D qui est décalé circulairement à gauche avec DF. Le bit à transmettre qui était dans DF vient ainsi prendre la place du bit de plus faible poids de D. Le nouveau contenu du registre D est restitué à R (D).0 qui se trouve ainsi lui-même décalé à gauche en ayant acquis à droite le bit en provenance de R (8).0.

Cette double opération est effectuée quatre fois (grâce au décomptage de R (9)).

d) *Stockage de la nouvelle donnée en mémoire*

Le registre R (A) contient l'adresse de la mémoire dont on est censé modifier le contenu. Comme la donnée modifiée est dans R (D).0, il suffit d'effectuer le transfert dans la mémoire, via l'accumulateur D.

On trouvera le programme correspondant à ce sous-ensemble aux adresses 00C5 à 00D1. Il se termine bien entendu par un branchement au sous-ensemble affichage-scrutation-encodage pour que l'on puisse afficher la nouvelle donnée et ordonner une nouvelle mission en appuyant sur une touche. Les deux sous-programmes se trouvent aux adresses 0149 à 014F et 0167 à 017A.

II. 7. Sous-ensemble Modification d'adresse

On y retrouve sensiblement les mêmes fonctions que précédemment, avec le stockage en mémoire en moins (voir organigrammes de la figure VIII. 10). Le décalage de l'espace d'affichage de l'adresse est un petit

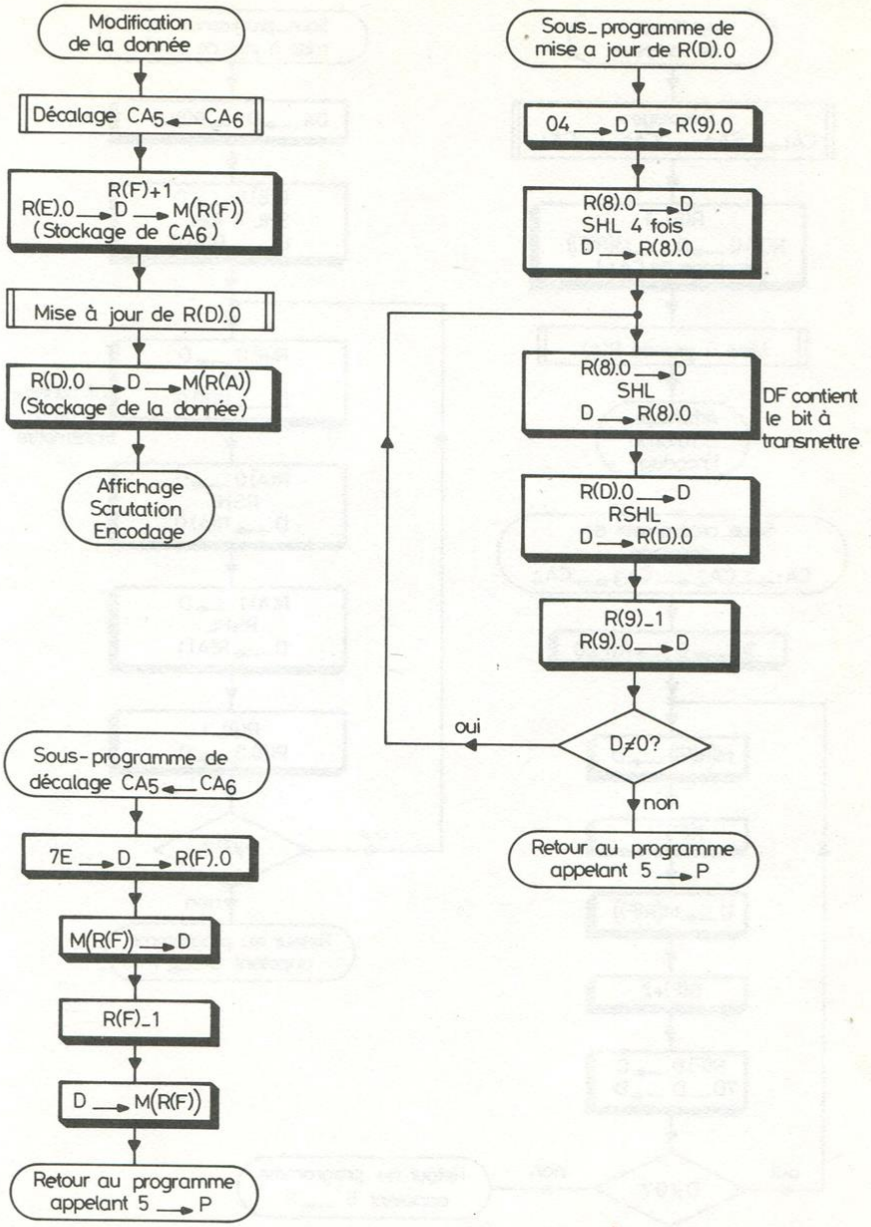


Fig. VIII. 9. Organigramme du sous-ensemble « modification de la donnée » et des sous-programmes utilisés

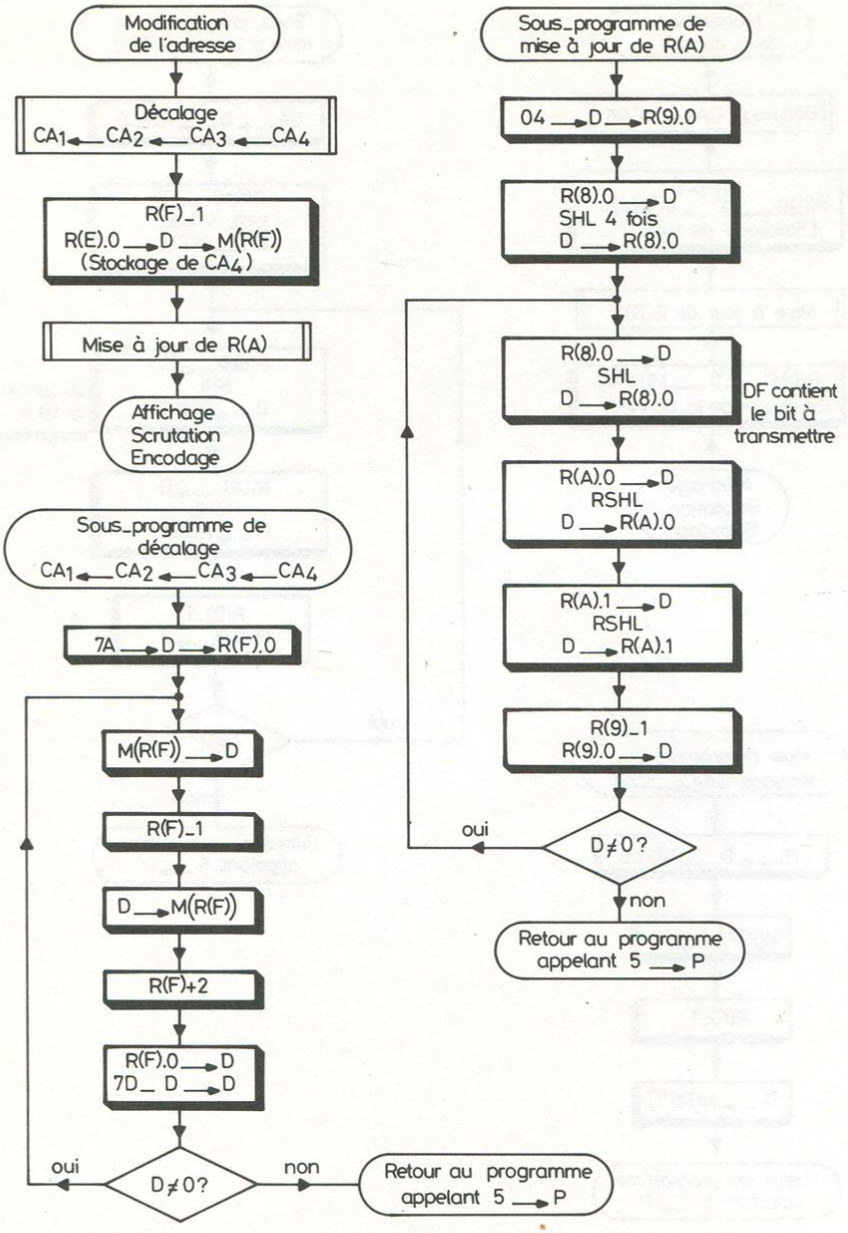


Fig. VIII. 10. Organigrammes du sous-ensemble « modification de l'adresse » et des sous-programmes utilisés

peu plus complexe car il porte maintenant sur quatre lignes au lieu de deux : il a été aventageux d'utiliser une structure de boucle. De la même manière la mise à jour du registre d'adresse R (A) se complique du fait de sa capacité de 16 bits au lieu des 8 bits du registre de donnée R (D).0. Néanmoins les méthodes utilisées restent les mêmes et nous laissons au lecteur le soin de décrypter seul les organigrammes de la figure VIII.10 en s'aidant éventuellement du programme écrit entre les adresses 00BA et 00C4 et des sous-programmes se trouvant aux adresses 013B à 0148 et 0150 à 0166.

II. 8. Sous-ensemble Identification de la fonction et branchement au programme correspondant

Dans le cas de la pression d'une touche de fonction, il faut, après la détermination de la nature de cette touche effectuée au paragraphe II 3, terminer le décodage pour aiguiller le programme vers l'exécution de cette fonction. Ceci est réalisé conformément à l'organigramme de la figure VIII. 10 bis. Souvenez-vous de la manière dont est formé le code d'une touche (chapitre 6) : le bit de fort poids B_7 de R (E).0 est mis à 1 si la touche U est utilisée ; c'est B_6 qui est mis à 1 si l'on utilise la touche P ; B_5 ou B_4 sont respectivement à 1 si la touche I ou M a été enfoncée.

Le code de la touche enfoncée (contenu dans R (E).0) est donc chargé dans D et l'on effectue un décalage à gauche dans le but d'amener B_7 sur le registre DF. Celui-ci est testé : s'il est à 1, c'est que la touche U a été utilisée et un branchement s'effectue au programme de la fonction réservée à l'utilisateur ; s'il est à 0, on recommence un décalage de D qui amène cette fois B_6 sur DF. Ce mécanisme reprend jusqu'à trouver le bit B_7, B_6, B_5 ou B_4 mis à 1 par l'enfoncement d'une touche de fonction, et brancher au programme correspondant. Le programme prévoit le branchement au sous-ensemble Affichage-Scrutation-Encodage au cas où aucun des quatre bits B_7, B_6, B_5 et B_4 ne serait à 1 : c'est qu'il se serait produit une anomalie de fonctionnement.

On trouvera le programme correspondant aux adresses 00D2 à 00E0.

III. Description des programmes des fonctions

III. 1. Fonction Examen de la mémoire (touche M)

La pression de la touche M doit conduire, rappelons-le, à l'affichage de la donnée écrite dans la mémoire à l'adresse indiquée sur les afficheurs d'adresse et contenue dans le registre R (A). Il faut pour cela lire la mémoire à l'adresse contenue dans R (A) et stocker dans l'espace d'affichage les codes d'affichage correspondants. Ceci est confié à un sous-programme spécifique (fig. VIII. 11). L'utilisation de la touche M doit être connue du

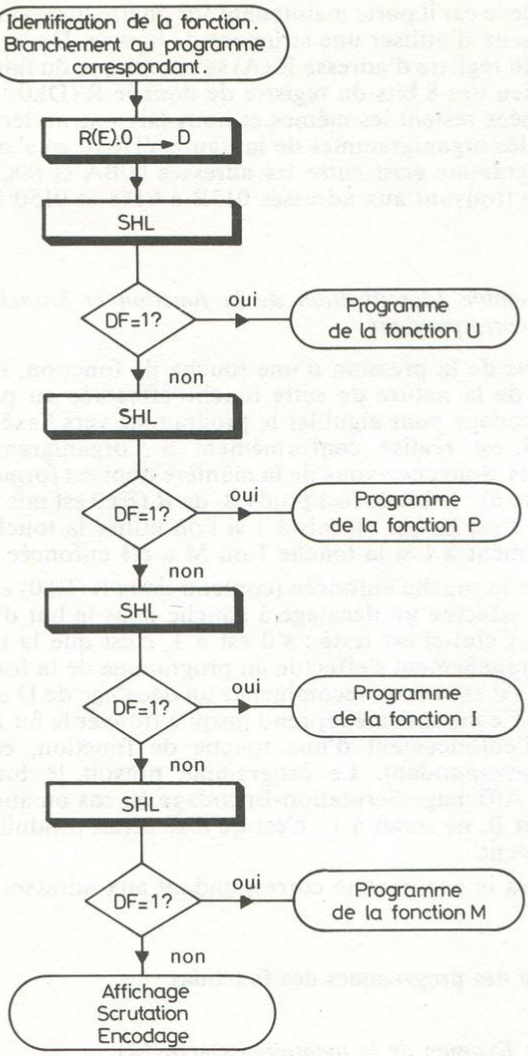


Fig. VIII.10 bis : Organigramme du sous-ensemble « identification de la fonction et branchement au programme correspondant »

moniteur ; aussi on a commencé par mettre l'indicateur IM à 1 en chargeant simplement 10000000 soit 80 en hexadécimal dans le registre R (E).1.

Le sous-programme de stockage des codes d'affichage de la donnée lue en mémoire procède en deux étapes :

a) L'octet VW est lu en mémoire à l'adresse contenue dans le registre R (A) et est chargé dans le registre de donnée R (D).0. On sélectionne le quartet de faible poids grâce à l'opération AND avec 00001111 (soit 0F en hexadécimal). Ce quartet correspondant au chiffre hexadécimal W, le registre D contient alors 0W. On y ajoute F0 pour former l'octet FW, poids faible de l'adresse où est inscrit le code d'affichage de W. Le registre R (8)

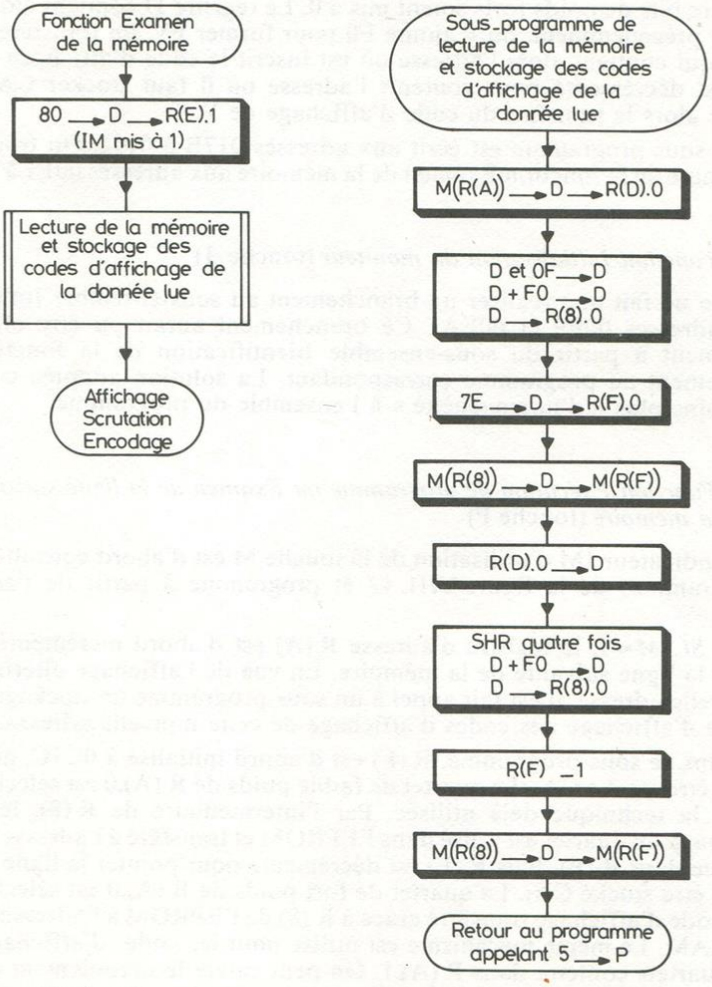


Fig. VIII. 11. Organigrammes de la fonction « examen de la mémoire » et du sous-programme utilisé

pointeur de cet espace est chargé par cette adresse alors que R (F) pointeur de l'espace d'affichage est chargé par l'adresse où il faut stocker CA_6. Tout est prêt pour le transfert du code d'affichage de W.

b) L'octet précédemment lu en mémoire et présent sur R (D).0 est à nouveau transféré sur l'accumulateur. Celui-ci subit alors quatre décalages à droite de manière que le quartet de poids fort (correspondant par exemple au chiffre hexadécimal V) prenne la place du quartet de poids fort et que les quatre bits de poids forts soient mis à 0. Le registre D contient alors 0V. Comme précédemment, on y ajoute F0 pour former FV, on transfère dans R (8).0 qui contient alors l'adresse où est inscrit le code d'affichage de V. R (F) est décrémenté pour contenir l'adresse où il faut stocker CA_5. On effectue alors le transfert du code d'affichage de V.

Ce sous-programme est écrit aux adresses 017B à 0192. On trouve le programme de la fonction Examen de la mémoire aux adresses 00E1 à 00E8.

III. 2. Fonction Initialisation du moniteur (touche I)

Elle ne fait que réaliser un branchement au sous-ensemble Initialisations (adresses 00E9 et 00EA). Ce branchement aurait pu être effectué directement à partir du sous-ensemble Identification de la fonction et branchement au programme correspondant. La solution adoptée confère néanmoins plus « d'homogénéité » à l'ensemble du programme.

III. 3. Fonction Exécution de programme ou Examen de la ligne suivante de la mémoire (touche P)

L'indicateur IM d'utilisation de la touche M est d'abord consulté (voir organigrammes de la figure VIII. 12 et programme à partir de l'adresse 00EB).

a) *Si IM = 1*, le registre d'adresse R (A) est d'abord incrémenté pour pointer la ligne suivante de la mémoire. En vue de l'affichage ultérieur de la nouvelle adresse, il est fait appel à un sous-programme de stockage dans l'espace d'affichage des codes d'affichage de cette nouvelle adresse.

Dans ce sous-programme, R (F) est d'abord initialisé à 0C7C, adresse où doit être stocké CA_4. Le quartet de faible poids de R (A).0 est sélectionné suivant la technique déjà utilisée. Par l'intermédiaire de R (8), le code d'affichage du quartet est puisé dans l'EPROM et transféré à l'adresse 0C7C contenue dans R (F). Puis R (F) est décrémenté pour pointer la ligne 0C7B où doit être stocké CA_3. Le quartet de fort poids de R (A).0 est sélectionné et son code d'affichage transféré grâce à R (8) de l'EPROM à l'adresse 0C7B de la RAM. Le même mécanisme est utilisé pour les codes d'affichage des deux quartets contenu dans R (A).1. On peut suivre le déroulement de ces opérations sur l'organigramme de la figure VIII.12 et aux lignes 0193 à 01BD du programme.

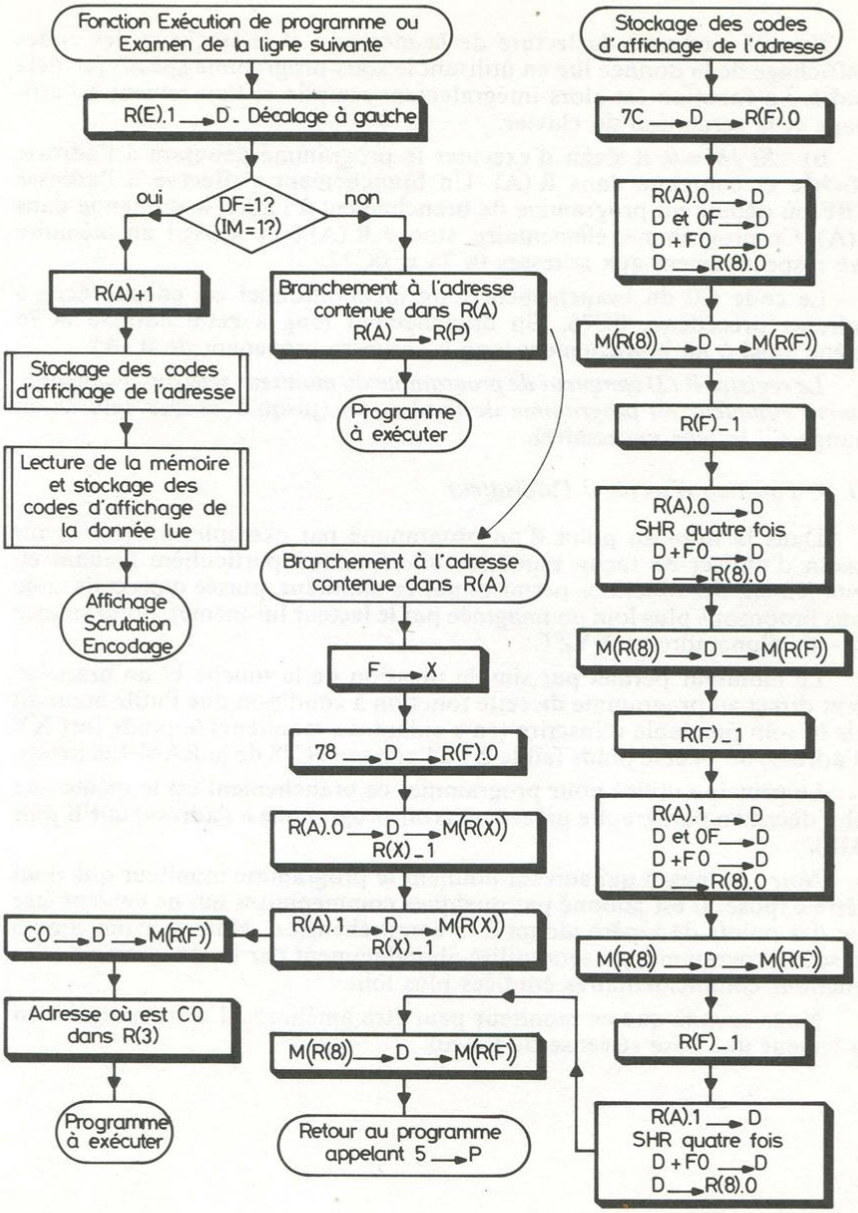


Fig. VIII. 12. Organigrammes de la fonction « exécution de programme » ou « examen de la ligne suivante » et du nouveau sous-programme utilisé

Ensuite s'effectue la lecture de la mémoire et le stockage des codes d'affichage de la donnée lue en utilisant le sous-programme spécifique déjà étudié. La fonction est alors intégralement remplie et l'on revient à l'affichage et la scrutation du clavier.

b) *Si IM = 0*, il s'agit d'exécuter le programme débutant à l'adresse affichée et contenue dans R (A). Un branchement s'effectue à l'adresse 01BE où débute un programme de branchement à l'adresse contenue dans R (A). Ce programme, élémentaire, stocke R (A).0 et R (A).1 en mémoire vive respectivement aux adresses 0C78 et 0C77.

Le code C0 du branchement long inconditionnel est ensuite écrit à l'adresse précédente 0C76. Un branchement long à cette adresse 0C76 amène ainsi à un branchement long à l'adresse provenant de R (A).

Le registre R (3) compteur de programme du moniteur sera aussi obligatoirement compteur du programme de l'utilisateur (jusqu'à ce que celui-ci en change s'il le juge nécessaire).

III. 4. Fonction réservée à l'utilisateur

Dans la mise au point d'un programme par exemple, on peut avoir besoin d'utiliser de façon répétitive une fonction particulière (venant en complément des fonctions permises par ce moniteur, puisée dans celles que nous proposons plus loin ou imaginée par le lecteur lui-même), programmée à partir d'une adresse XYZT.

Le moniteur permet par simple pression de la touche U un branchement direct au programme de cette fonction à condition que l'utilisateur ait pris le soin préalable d'inscrire (en s'aidant du moniteur) le poids fort XY à l'adresse 0C74 et le poids faible ZT à l'adresse 0C75 de la RAM 128 octets.

Le principe utilisé pour programmer ce branchement est le même que celui décrit au paragraphe précédent (voir programme à l'adresse 00FB puis 0318).

Note : les pages qui suivent donnent le programme moniteur qui vient d'être exposé. Il est jalonné par quelques commentaires qui ne veulent être que des points de repère aidant à la compréhension. Nous y avons ajouté un sous-programme qui sera utilisé abondamment par les programmes des fonctions complémentaires étudiées plus loin.

Nous savons que ce moniteur peut être amélioré. Il offre malgré tout au lecteur une base sérieuse de travail.

PROGRAMME MONITEUR

Initialisations générales : Compteur de programme : registre R(0)

Adresses	Codes et opérandes	Mnémoniques	Commentaires
0000	F8 00	LDI 00	
02	B3	PHI R(3)	
03	B4	PHI R(4)	
04	B5	PHI R(5)	
05	F8 01	LDI 01	
07	B8	PHI R(8)	
08	F8 0C	LDI 0C	
0A	B2	PHI R(2)	
0B	BF	PHI R(F)	
0C	F8 37	LDI 37	
0E	A3	PLO R(3)	
0F	F8 1B	LDI 1B	
11	A4	PLO R(4)	
12	F8 2B	LDI 2B	
14	A5	PLO R(5)	
15	F8 72	LDI 72	
17	A2	PLO R(2)	
18	7A	REQ	
19	D3	SEP R(3)	Appel du programme principal

Sous-programme d'appel d'un sous-programme standard

Compteur de programme : registre R(4)

←001A	D3	SEP R(3)	Renvoi au sous-programme appelé
→ 1B	E2	SEX R(2)	
1C	96	GHI R(6)	
1D	73	STXD	
1E	86	GLO R(6)	
1F	73	STXD	
20	93	GHI R(3)	
21	B6	PHI R(6)	
22	83	GLO R(3)	
23	A6	PLO R(6)	
24	46	LDA R(6)	
25	B3	PHI R(3)	
26	46	LDA R(6)	
27	A3	PLO R(3)	
28	30 1A	BR	Branchement à 1A, sortie du sous-programme

Sous-programme de retour au programme appelant

Compteur de programme : registre R(5)

002A	D3	SEP R(3)	Renvoi au programme appelant
2B	96	GHI R(6)	
2C	B3	PHI R(3)	
2D	86	GLO R(6)	
2E	A3	PLO R(3)	
2F	E2	SEX R(2)	
30	12	INC R(2)	
31	72	LDXA	
32	A6	PLO R(6)	
33	F0	LDX	
34	B6	PHI R(6)	
35	30 2A	BR	Branchement à 2A, sortie du sous-programme

Programme principal : Compteur de programme : registre R(3)

1. Initialisations (suite)

– du PPI 8255

0037	F8 20	LDI 20	
39	B7	PHI R(7)	
3A	F8 03	LDI 03	
3C	A7	PLO R(7)	R(7) pointe le registre de contrôle du 8255
3D	E7	SEX R(7)	
3E	F8 82	LDI 82	
40	73	STXD	Ports A et C en sortie, B en entrée
41	27	DEC R(7)	
42	27	DEC R(7)	R(7) pointe le port A
43	F8 04	LDI 04	
45	57	STR R(7)	$PA_2 = \overline{OE}$ mis à 1 - $PA_5 = \overline{CE}$ mis à 0
46	17	INC R(7)	
47	17	INC R(7)	R(7) pointe le port C

– de l'espace d'affichage (en vue de l'affichage de PrEt)

0048	F8 7F	LDI 7F	
4A	AF	PLO R(F)	R(F) pointe le bas de l'espace d'affichage
4B	EF	SEX R(F)	
4C	F8 FF	LDI FF	Code extinction
4E	73	STXD	
4F	73	STXD	CA_6 = FF
50	73	STXD	CA_5 = FF
51	F8 3C	LDI 3C	Code d'affichage de t

```
53      73          STXD        CA4 = 3C
54      F8 34       LDI 34      Code d'affichage de E
56      73          STXD        CA3 = 34
57      F8 7E       LDI 7E      Code d'affichage de r
59      73          STXD        CA2 = 7E
5A      F8 70       LDI 70      Code d'affichage de P
5C      5F          STR R(F)    CA1 = 70
```

– de l'indicateur IM d'utilisation de la touche M

```
005D    F8 00       LDI 00
5F      BE          PHI R(E)
```

2. Affichage - Scrutation et encodage du clavier

– affichage, scrutation, traitement du rebondissement (détection de l'enfoncement d'une touche)

```
0060    D4 010D     SEP R(4)    Appel du SP d'affichage
63      D4 0125     SEP R(4)    Appel du SP de scrutation
66      99          GHI R(9)
67      32 60       BZ          Retour à affichage si pas de touche enfoncée
69      D4 0100     SEP R(4)    Appel du SP de temporisation...   } Anti-
            0341                ... de 20 ms                      } rebond
6E      D4 0125     SEP R(4)    Appel du SP de scrutation         } Anti-
71      99          GHI R(9)                                      } rebond
72      32 60       BZ          Retour à affichage si pas de touche enfoncée } Anti-rebond
```

– encodage dans R(E).0 de la touche enfoncée

```
0074    99          GHI R(9)
75      AE          PLO R(9)    Le code de la ligne est dans R(E) 0
76      64          OUT 4       Initialisation du 4022
77      22          DEC R(2)
78      62          OUT 2       Incrémentation du 4022
79      22          DEC R (2)
7A      34 84       B1          Branchement si EF1 = 0
7C      35 84       B2          Branchement si EF2 = 0
7E      36 84       B3          Branchement si EF3 = 0
80      37 84       B4          Branchement si EF4 = 0
82      30 8C       BR          Branchement à détection du relâchement de la touche
84      8E          GLO R(E)
85      FA 04       ANI 04
87      3A 60       BNZ         Branchement à affichage scrutation car anomalie
89      1E          INC R(E)
8A      30 78       BR          Branchement pour incrémentation du 4022
```

– affichage, scrutation, traitement du rebondissement (détection du relâchement de la touche)

```
008C   D4 010D   SEP R(4)   Appel du S.P. d'affichage
  8F   D4 0125   SEP R(4)   Appel du S.P. de scrutation
  92   99        GHI R(9)
  93   3A 8C     BNZ       Retour à affichage si touche encore enfoncée
  95   D4 0100   SEP R(4)   Appel du S.P. de temporisation...   }
       0341                 ... de 20 ms                       }
  9A   D4 0125   SEP R(4)   Appel du S.P. de scrutation         } Anti-
  9D   99        GHI R(9)                                      } rebond
  9E   3A 8C     BNZ       Retour à affichage si touche encore enfoncée }
```

3. Détermination de la nature de la touche enfoncée

```
00A0   8E        GLO R(E)
  A1   FA 04     ANI 04
  A3   3A D2     BNZ       Branchement à l'identification de la fonction
                           si c'est une touche de fonction
```

4. Décodage d'une touche de caractère hexadécimal et formation du code d'affichage de ce caractère dans R(E).0

```
00A5   F8 E0     LDI E0
  A7   A8        PLO R(8)  – R(8) contient 01E0 (adresse du haut de l'espace des codes)
  A8   E8        SEX R(8)
  A9   8E        GLO R(E)
  AA   F7        SM
  AB   32 B0     BZ
  AD   18        INC R(8)
  AE   30 A9     BR
  B0   88        GLO R(8)
  B1   FC 10     ADI
  B3   A8        PLO R(8)  R(8) pointe le code d'affichage
  B4   F0        LDX
  B5   AE        PLO R(E)  R(E) contient le code d'affichage
```

5. Test de l'indicateur IM d'utilisation de la touche M

```
00B6   9E        GHI R(E)
  B7   FE        SHL
  B8   33 C5     BDF       Branchement à modification de donnée si IM = 1
```

6. Modification de l'adresse

```
00BA   D4 013B   SEP R(4)   Appel du S.P. de décalage C1₁ ← CA₂ ← CA₃ ← CA₄
  BD   2F        DEC R(F)
```

```
BE      8E          GLO R(E)
BF      5F          STR R(F)    Code d'affichage de la touche enfoncée = CA4
C0      D4 0150     SEP R(4)    Appel du S.P. de mise à jour du registre d'adresse R(A)
C3      30 60       BR          Retour à affichage - scrutation et encodage du clavier
```

7. Modification de la donnée

```
00C5    D4 0149     SEP R(4)    Appel du S.P. de décalage CA5 ← CA6
C8      1F          INC R(F)
C9      8E          GLO R(E)
CA      5F          STR R(F)    Code d'affichage de la touche enfoncée = CA6
CB      D4 0167     SEP R(4)    Appel du S.P. mise à jour du registre de donnée R(D).0
CE      8D          GLO R(D)
CF      5A          STR R(A)    Nouvelle donnée stockée dans RAM
D0      30 60       BR          Retour à affichage - scrutation et encodage du clavier
```

9. Identification de la fonction et branchement au programme correspondant

```
00D2    8E          GLO R(E)
D3      FE          SHL
D4      33 FB       BDF         Branchement au programme de la fonction U
D6      FE          SHL
D7      33 EB       BDF         Branchement au programme de la fonction P
D9      FE          SHL
DA      33 E9       BDF         Branchement au programme de la fonction I
DC      FE          SHL
DD      33 E1       BDF         Branchement au programme de la fonction M
DF      30 60       BR          Retour à affichage-scrutation car anomalie
```

Programme des fonctions : Compteur de programme : registre R(3)

1. Fonction examen de la mémoire (touche M)

```
00E1    F8 80       LDI 80
E3      BE          PHI R(E)    IM mis à 1
E4      D4 017B     SEP R(4)    Appel du S.P. lecture de la mémoire et stockage
                                des codes d'affichage de la donnée
E7      30 60       BR          Retour à affichage-scrutation et encodage du clavier
```

2. Fonction initialisation

```
00E9    30 37       BR          Retour à initialisations
```

3. Fonction incrémentation de l'adresse ou exécution d'un programme

```
00EB    9E          GHI R(E)
```

EC	FE	SHL	
ED	33 F2	BDF	
EF	C0 01BE	LBR	Branchement au programme de branchement à l'adresse R(A)
F2	1A	INC R(A)	
F3	D4 0193	SEP R(4)	Appel du S.P. de stockage des codes d'affichage de l'adresse
F6	D4 017B	SEP R(4)	Appel du S.P. de lecture de la mémoire et stockage des codes d'affichage de la donnée
F9	30 60	BR	Retour à affichage-scrutation et encodage du clavier

4. Fonction utilisateur

00FB	C0 0328	LBR	Le programme est écrit à partir de l'adresse 0328

Sous-programmes standards : Compteur de programme : registre R(3)

1. Temporisation

0100	46	LDA R(6)	
01	BC	PHI R(C)	
02	46	LDA R(6)	
03	AC	PLO R(C)	Le compteur de boucle est chargé
04	2C	DEC R(C)	
05	9C	GHI R(C)	
06	3A 04	BNZ	
08	2C	DEC R(C)	
09	8C	GLO R(C)	
0A	3A 08	BNZ	
0C	D5	SEP R(5)	Retour au programme appelant.

2. Affichage

010D	F8 78	LDI 78	
0F	AF	PLO R(F)	R(F) contient l'adresse 0C78
10	F8 06	LDI 06	
12	A9	PLO R(9)	R(9).0 compteur de boucle contient 06
13	EF	SEX R(F)	
14	64	OUT4	Initialisation du 4022. R(F) contient 0C79
15	F0	LDX	
16	57	STR R(7)	Code d'affichage sur port C
17	62	OUT2	Incrémentation du 4022 et de R(F)
18	D4 0100 002A	SEP R(4)	Appel du S.P. de temporisation... ... de 1 ms
1D	EF	SEX R(F)	
1E	29	DEC R(9)	

```
1F    89       GLO R (9)
20    3A 15    BNZ
22    F0       LDX
23    57       STR R(7)
24    D5       SEP R(5)      Retour au programme appelant
```

3. Scrutation du clavier et formation du code de la ligne dans R(9).1

```
0125  EF       SEX R(F)
26    64       OUT4          Initialisation du 4022
27    2F       DEC R(F)
28    F8 10    LDI 10
2A    34 39    B1
2C    35 38    B2
2E    36 37    B3
30    37 36    B4
32    F8 00    LDI 00
34    30 39    BR
36    FE       SHL
37    FE       SHL
38    FE       SHL
39    B9       PHI R(9)      Sauvegarde du code de la ligne
3A    D5       SEP R(5)      Retour au programme appelant
```

4. Décalage $CA_1 \leftarrow CA_2 \leftarrow CA_3 \leftarrow CA_4$

```
013B  F8 7A    LDI 7A
3D    AF       PLO R(F)      R(F) contient l'adresse où est CA2
3E    0F       LDN R(F)
3F    2F       DEC R(F)      R(F) contient l'adresse où le code doit être stocké
40    5F       STR R(F)
41    1F       INC R(F)
42    1F       INC R(F)      R(F) contient l'adresse du nouveau code à transférer
43    8F       GLO R(F)
44    FD 7D    SDI           Comparaison de R(F) avec 7D pour savoir si décalage
                             terminé
46    3A 3E    BNZ
48    D5       SEP R(5)      Retour au programme appelant
```

5. Décalage $CA_5 \leftarrow CA_6$

```
0149  F8 7E    LDI 7E
4B    AF       PLO R(F)      R(F) contient l'adresse où est CA6
4C    0F       LDN R(F)
```

```
4D    2F      DEC R(F)    R(F) contient l'adresse où le code doit être stocké
4E    5F      STR R(F)
4F    D5      SEP R(5)    Retour au programme appelant
```

6. Mise à jour du registre d'adresse R(A)

```
0150  F8 04   LDI 04
52    A9      PLO R(9)    R(9) contient le nombre de décalages à effectuer
53    88      GLO R(8)
54    7E      RSHL
55    7E      RSHL
56    7E      RSHL
57    7E      RSHL
58    A8      PLO R(8)    Les bits à transférer sont à gauche sur R(8).0
59    88      GLO R(8)
5A    7E      RSHL        MSB de R(8).0 dans DF
5B    A8      PLO R(8)    et R(8).0 décalé à gauche
5C    8A      GLO R(A)
5D    7E      RSHL        MSB de R(A).0 dans DF et ex MSB de R(8).0 à droite dans D
5E    AA      PLO R(A)    R(A).0 décalé à gauche et ex MSB de R(8).0 à droite dans R(A).0
5F    9A      GHI R(A)
60    7E      RSHL        ex MSB de R(A).0 à droite dans D
61    BA      PHI R(A)    R(A).1 décalé à gauche et ex MSB de R(A).0 à droite dans R(A).1
62    29      DEC R(9)
63    89      PLO R(9)
64    3A 59   BNZ
66    D5      SEP R(5)    Retour au programme appelant
```

7. Mise à jour du registre de donnée R(D).0

```
0167  F8 04   LDI 04
69    A9      PLO R(9)
6A    88      GLO R(8)
6B    7E      RSHL
6C    7E      RSHL
6D    7E      RSHL
6E    7E      RSHL
6F    A8      PLO R(8)
70    88      GLO R(8)
71    7E      RSHL        MSB de R(8).0 dans DF
72    A8      PLO R(8)    et R(8).0 décalé à gauche
```

73	8D	GLO R(D)	
74	7E	RSHL	ex MSB de R(8).0 à droite dans D
75	AD	PLO R(D)	R(D).0 décalé à gauche et ex MSB de R(8).0 à droite dans R(D).0
76	29	DEC R(9)	
77	89	GLO R(9)	
78	3A 70	BNZ	
7A	D5	SEP R(5)	Retour au programme appelant

8. Lecture de la mémoire et stockage des codes d'affichage de la donnée lue :

017B	0A	LDN R(A)	Donnée dans D...
7C	AD	PLO R(D)	... et dans R(D).0
7D	FA 0F	ANI 0F	Sélection du quartet de faible poids
7F	FC F0	ADI F0	
81	A8	PLO R(8)	R(8) contient l'adresse où est situé le code d'affichage
82	F8 7E	LDI 7E	
84	AF	PLO R(F)	R(F) contient l'adresse où stocker CA_6
85	08	LDN R(8)	
86	5F	STR R(F)	CA_6 stocké
87	8D	GLO R(D)	
88	F6	SHR	}
89	F6	SHR	} Sélection du quartet de poids fort
8A	F6	SHR	}
8B	F6	SHR	}
8C	FC F0	ADI F0	
8E	A8	PLO R(8)	R(8) contient l'adresse où est situé le code d'affichage
8F	2F	DEC R(F)	R(F) contient l'adresse où stocker CA_5
90	08	LDN R(8)	
91	5F	STR R(F)	CA_5 stocké
92	D5	SEP R(5)	Retour au programme appelant

9. Stockage des codes d'affichage de l'adresse contenue dans R(A)

0193	F8 7C	LDI 7C	
95	AF	PLO R(F)	R(F) contient l'adresse où stocker CA_4
96	8A	GLO R(A)	
97	FA 0F	ANI 0F	Sélection du quartet de faible poids de R(A).0
99	FC F0	ADI F0	
9B	A8	PLO R(8)	R(8) contient l'adresse où est situé le code d'affichage
9C	08	LDN R(8)	
9D	5F	STR R(F)	CA_4 stocké
9E	2F	DEC R(F)	R(F) contient l'adresse où stocker CA_3

```
9F   8A      GLO R(A)
A0   F6      SHR  ┐
A1   F6      SHR  │
A2   F6      SHR  │  Sélection du quartet de fort poids de R(A).0
A3   F6      SHR  ┘
A4   FC F0   ADI F0
A6   A8      PLO R(8)    R(8) contient l'adresse où est situé le code d'affichage
A7   08      LDN R(8)
A8   5F      STR R(F)    CA3 stocké
A9   2F      DEC R(F)    R(F) contient l'adresse où stocker CA2
AA   9A      GHI R(A)
AB   FA 0F   ANI 0F      Sélection du quartet de faible poids de R(A).1
AD   FC F0   ADI F0
AF   A8      PLO R(8)
B0   08      LDN R(8)
B1   5F      STR R(F)    CA2 stocké
B2   2F      DEC R(F)    R(F) contient l'adresse où stocker CA1
B3   9A      GHI R(A)
B4   F6      SHR  ┐
B5   F6      SHR  │  Sélection du quartet et de fort poids de R(A).1
B6   F6      SHR  │
B7   F6      SHR  ┘
B8   FC F0   ADI F0
BA   A8      PLO R(8)
BB   08      LDN R(8)
BC   5F      STR R(F)    CA1 stocké
BD   D5      SEP R(5)    Retour au programme appelant
```

*** Branchement du programme à l'adresse contenue dans R(A)**
N.B. : Ce n'est pas un sous-programme mais une partie du programme.

```
01BE   EF       SEX R(F)
BF     F8 78    LDI 78
C1     AF       PLO R(F)    Initialisation de R(F) à 0C78
C2     8A       GLO R(A)
C3     73       STXD        M (0C78) contient R(A).0 et R(F) contient 0C77
C4     9A       GHI R(A)
C5     73       STXD        M (0C77) contient R(A).1 et R(F) contient 0C76
C6     F8 C0    LDI C0
C8     73       STXD        M (0C76) contient C0, code de LBR
C9     C0 0C76  LBR         vers demande de branchement
```

8 bis. Variante du sous-programme 8 (pas de lecture de la mémoire) utilisé pour lecture d'EPROM

01CC	8D	GLO R(D)	
CD	30 7D	BR	Branchement au stockage des codes d'affichage de la donnée

10. Chargement des registres R(A), R(B), R(1) (utilisé pour fonctions complémentaires)

01CF	F8 00	LDI 00	
D1	AF	PLO R(F)	R(F) initialisé à 0C00
D2	EF	SEX R(F)	
D3	72	LDXA	
D4	BA	PHI R(A)	R(A).1 contient M (0C00)
D5	72	LDXA	
D6	AA	PLO R(A)	R(A).0 contient M (0C01)
D7	72	LDXA	
D8	BB	PHI R(B)	R(B).1 contient M (0C02)
D9	72	LDXA	
DA	AB	PLO R(B)	R(B).0 contient M (0C03)
DB	72	LDXA	
DC	B1	PHI R(1)	R(1).1 contient M (0C04)
DD	72	LDXA	
DE	A1	PLO R(1)	R(1).0 contient M (0C05)
DF	D5	SEP R(5)	Retour au programme appelant

Codes des touches hexadécimales

01E0	10
E1	11
E2	12
E3	13
E4	20
E5	21
E6	22
E7	23
E8	40
E9	41
EA	42
EB	43
EC	80
ED	81
EE	82
EF	83

Codes d'affichage des caractères hexadécimaux

01F0	21
F1	EB
F2	32
F3	A2
F4	E8
F5	A4
F6	24
F7	E3
F8	20
F9	E0
FA	60
FB	2C
FC	35
FD	2A
FE	34
FF	74

Fin du programme de la fonction utilisateur

```
0328    F8 73       LDI 73
 2A     AF          PLO R(F)    R(F) contient 0C73
 2B     F8 C0       LDI C0
 2D     5F          STR R(F)    M(0C73) contient C0, code de LBR
 2E     C0 0C73     LBR         Vers demande de branchement
```

Chapitre 9

PROGRAMMATEUR D'EPROM

Les programmes et certaines listes de données de la plupart des applications un peu élaborées de la microélectronique programmée sont figés dans des mémoires à lecture seule (ROM). L'intérêt fondamental de ces mémoires est que, à l'opposé des mémoires à lecture et à écriture (RAM), elles conservent leur contenu même en l'absence d'alimentation.

Le développement d'un projet utilisant un microprocesseur passe nécessairement par la phase d'étude et de mise au point du programme. Dans cette étape, on utilise uniquement une mémoire RAM qui autorise écritures et modifications, décalages, vérifications..., opérations facilitées par l'emploi d'un moniteur. Quand la mise au point est terminée, il faut transférer le programme définitif dans une ROM. Cette opération n'est possible que si la ROM est *programmable* (PROM). Il existe sur le marché des appareils — programmateurs de PROM — remplissant cette fonction ; destinés à l'industrie ou à la recherche, leur prix rebute souvent l'amateur peu fortuné.

Le système VILÉMIO est conçu pour que l'on puisse charger (ou *programmer*) une EPROM de type 2716, *en entier ou par fragments*. Quand on a couplé les maquettes 1, 2 et 3, on peut écrire *où l'on veut* dans l'EPROM à programmer, une série d'octets provenant :

— soit de la RAM 2 K octets : il s'agira généralement d'un programme que l'on vient de mettre au point ;

— soit de l'EPROM 2 K octets située sur la maquette processeur : il s'agit alors de recopier une EPROM ;

— soit encore de l'EPROM moniteur (duplication du moniteur) ou même de la RAM 128 octets de la maquette processeur.

Le programme de programmation d'EPROM 2716 est bien entendu inscrit dans l'EPROM moniteur. Il est complété par un programme qui permet la vérification de l'état de virginité de la zone que l'on désire programmer ou bien la lecture et l'affichage du contenu des lignes successives de toute une zone.

On se propose ici d'indiquer la structure matérielle retenue, d'expliquer les programmes de programmation et de lecture d'EPROM, puis d'exposer la procédure d'emploi.

I. Aspect matériel

I. 1 L'EPROM type 2716

C'est une mémoire à lecture seule, programmable électriquement, effaçable par rayonnement ultraviolet. Parmi d'autres avantages, elle offre celui de ne nécessiter qu'une alimentation (5 V) en mode lecture.

La capacité de l'EPROM 2716 est de 2 K octets. Huit broches du boîtier sont donc consacrées à la sortie des données (O_0 à O_7) et onze à l'adressage (A_0 à A_{10}). Deux broches sont nécessaires pour l'alimentation. Les trois dernières reçoivent respectivement les signaux de contrôle $\overline{CE}$/PGM (chip enable/program) et $\overline{OE}$ (output enable) et la tension $V_{pp} = 25 \pm 1$ V nécessaire à la programmation. Un schéma de brochage est fourni par la figure II. 3.

I. 2. Liaison de l'EPROM avec le reste du système

L'EPROM 2716 à programmer ou lire se place sur un support à force d'insertion nulle. Elle se trouve alors alimentée sous 5 V. Une commande par inverseur permet d'appliquer sur la broche V_{pp} soit les 25 V nécessaires à la programmation provenant d'une source extérieure reliée aux cosses notées « 25 V, masse » (position « oui » de l'inverseur), soit les 5 V requis pour un fonctionnement normal en lecture (position « non » de l'inverseur).

Il faut bien comprendre que l'EPROM à programmer ou lire ne fait pas partie de l'ensemble de mémoire directement adressable par le microprocesseur. Elle occupe ici la place d'un dispositif extérieur à commander par le système microprocesseur-mémoire associée auquel elle est reliée par des ports d'entrée/sortie. Les données à programmer ou à lire dans l'EPROM s'échangent avec le système microprocesseur-mémoire via le port B du PPI 8255 programmé suivant le cas soit en sortie, soit en entrée. Le système fournira les adresses de l'EPROM ainsi que les signaux de contrôle $\overline{CE}$/PGM et $\overline{OE}$ par l'intermédiaire des ports A et C. La figure IX. 1 indique les liaisons existant entre les ports du 8255 et l'EPROM 2716. On note que l'ordre des lignes d'adresse ne suit pas l'ordre des broches des ports A et C du 8255. Cette correspondance imposée par la conception du circuit imprimé aura pour conséquence de compliquer un peu la programmation. On se heurte une nouvelle fois au dilemme matériel-logiciel : la simplification de l'un conduit à une complication de l'autre ; il faut choisir !

II. Programmation de l'EPROM 2716

II. 1. Les principes

La consultation attentive des notices fournies par les fabricants

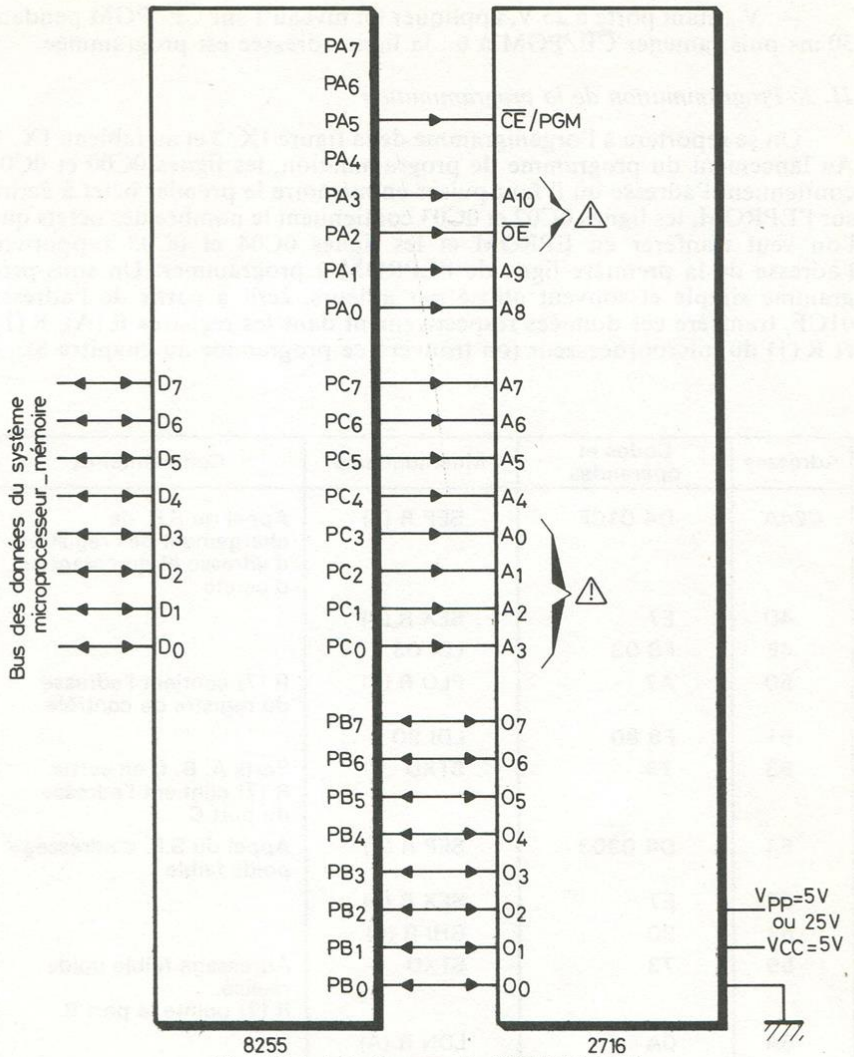


Fig. IX. 1. Interfaçage de l'EPROM 2716

d'EPROM 2716 ou équivalentes conduit à adopter le processus suivant pour la programmation d'une ligne :

— $\overline{OE}$ étant au niveau logique 1 (ce qui autorise les lignes O_0 à O_7 de la 2716 à recevoir une donnée) et $\overline{CE}$/PGM à 0, présenter la donnée à écrire sur les lignes O_0 à O_7 et l'adresse sur les lignes A_0 à A_{10} ;

— V_{pp} étant porté à 25 V, appliquer un niveau 1 sur $\overline{CE}$/PGM pendant 50 ms puis ramener $\overline{CE}$/PGM à 0 : la ligne adressée est programmée.

II. 2. Programmation de la programmation

On se reportera à l'organigramme de la figure IX. 2 et au tableau IX. 1. Au lancement du programme de programmation, les lignes 0C00 et 0C01 contiennent l'adresse où il faut puiser en mémoire le premier octet à écrire sur l'EPROM, les lignes 0C02 et 0C03 contiennent le nombre des octets que l'on veut tranférer en EPROM et les lignes 0C04 et 0C05 supportent l'adresse de la première ligne de l'EPROM à programmer. Un sous-programme simple et souvent utilisé par ailleurs, écrit à partir de l'adresse 01CF, transfère ces données respectivement dans les registres R (A), R (B) et R (1) du microprocesseur (on trouvera ce programme au chapitre 8).

Adresses	Codes et opérandes	Mnémoniques	Commentaires
024A	D4 01CF	SEP R (4)	Appel du S.P. de chargement des registres d'adresse et du compteur d'octets
4D	E7	SEX R (7)	
4E	F8 03	LDI 03	
50	A7	PLO R (7)	R (7) contient l'adresse du registre de contrôle
51	F8 80	LDI 80	
53	73	STXD	Ports A, B, C en sortie. R (7) contient l'adresse du port C
54	D4 0303	SEP R (4)	Appel du S.P. d'adressage poids faible
57	E7	SEX R (7)	
58	90	GHI R (0)	
59	73	STXD	Adressage faible poids réalisé. R (7) pointe le port B.
5A	0A	LDN R (A)	
5B	73	STXD	Donnée à écrire présentée à l'EPROM. R (7) pointe le port A.
5C	91	GHI R (1)	
5D	FC 04	ADI 04	
5F	F9 04	ORI 04	

Adresses	Codes et opérandes	Mnémoniques	Commentaires
61	57	STR R (7)	Adressage fort poids réalisé - $\overline{CE}$ = 0 - $\overline{OE}$ = 1
62	FC 20	ADI	
64	57	STR R (7)	$\overline{CE}$ mis à 1
65	AE	PLO R (E)	Sauvegarde
66	D4 0100 0823	SEP R (4)	Appel du S.P. de temporisation... de 50 ms
6B	8E	GLO R (E)	
6C	FF 20	SMI	
6E	57	STR R (7)	$\overline{CE}$ remis à 0
6F	1A	INC R (A)	
70	11	INC R (1)	
71	2B	DEC R (B)	
72	9B	GHI R (B)	Test de R (B). Branchement pour programmer une nouvelle ligne si liste non terminée.
73	3A 4D	BNZ 4D	
75	8B	GLO R (B)	
76	3A 4D	GNZ 4D	
78	C0 0037	LBR	Retour au moniteur

Tableau IX.1. Programme de programmation d'EPROM

Le registre R (7) est utilisé pour pointer les quatre registres du PPI 8255. Comme l'on fait souvent appel à ses services, on le désigne par le registre X en chargeant 7 dans X.

La programmation en sortie des trois ports A, B, C s'effectue en stockant le code 80 dans le registre de contrôle du 8255, situé à l'adresse 2003. Cette opération exige le chargement préalable du poids faible 03 dans R (7).0 (le poids fort 20 a été chargé dans R (7).1 à l'initialisation du moniteur).

L'adressage de poids faible de l'EPROM consiste à appliquer R (1).0 sur les lignes A_0 à A_7. Cette opération serait simple s'il y avait correspondance directe entre les broches A_0 à A_7 et les broches PC_0 à PC_7. Il suffirait alors de stocker R (1).0 sur le port C grâce aux opérations successives R (1).0 → D et D → M (2002). Les permutations $PC_3 - PC_0$ et $PC_2 - PC_1$ obligent à une manipulation préalable confiée au sous-programme d'adresse 0303. Par le biais de décalages et de transferts successifs, ce sous-programme qui utilise R (0).0 comme registre de sauvegarde forme dans R (0).1 le mot binaire qui, stocké sur le port C, réalisera (A_7 --- A_0) = R (1).0. Il est fourni au tableau 2. Nous laissons au lecteur le soin de le déchiffrer.

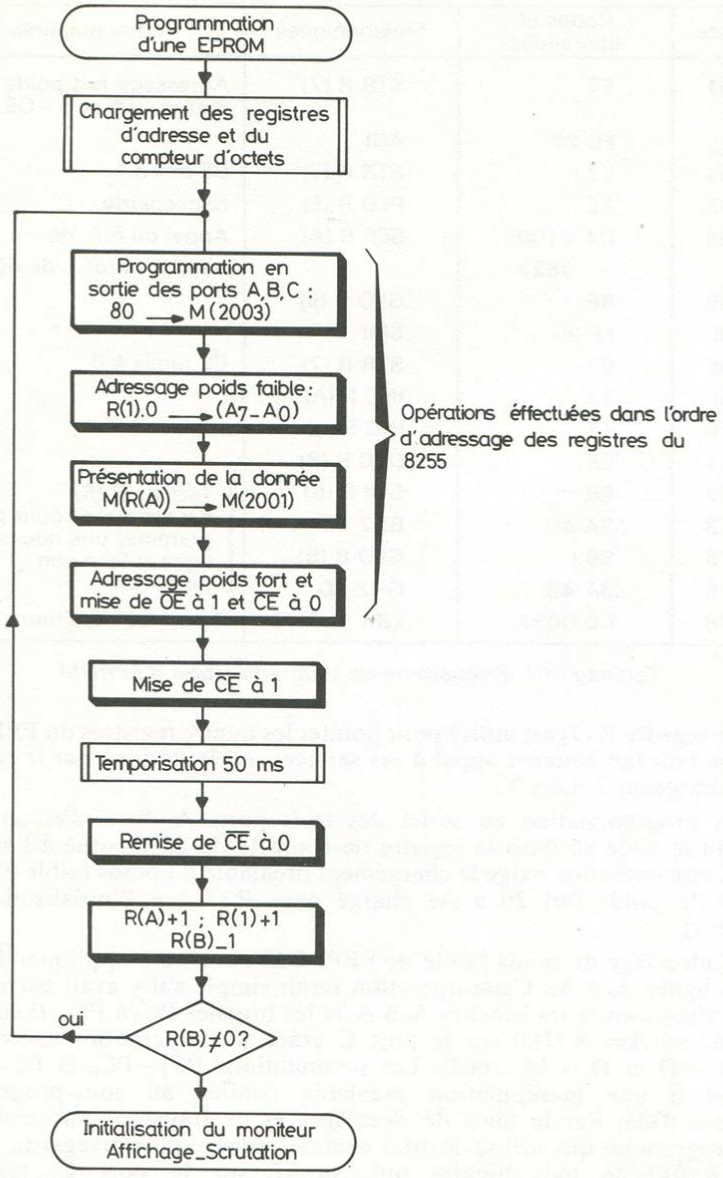


Fig. IX. 2. Organigramme du programme de programmation d'EPROM

Adresses	Codes et opérandes	Mnémoniques	Commentaires
0303	81	GLO R (1)	D contient poids faible $A_7 \ldots A_0$.
04	A0	PLO R (0)	R (0). 0 contient le poids faible de l'adresse.
05	F6	SHR	
06	F6	SHR	
07	F6	SHR	
08	F6	SHR	
09	B0	PHI R (0)	R (0). 1 contient $0000A_7A_6A_5A_4$.
0A	F8 04	LDI 04	
0C	A9	PLO R (9)	Compteur de boucles R (9) chargé par 04.
0D	80	GLO R (0)	R (0). décalé à droite. DF contient A_0 puis A_1, A_2, A_3
0E	F6	SHR	
0F	A0	PLO R (0)	
10	90	GHI R (0)	R (0). 1 décalé à gauche DF = A_0 puis A_1, A_2, A_3 chargé à droite.
11	7E	RSHL	
12	B0	PHI R (0)	
13	29	DEC R (9)	Décomptage et branchement pour effectuer 4 fois les deux opérations précédentes.
14	89	GLO R (9)	
15	3A 0D	BNZ 0D	
17	D5	SEP R (5)	Retour au programme appelant.

Tableau IX. 2. Sous-programme de formation de l'octet de poids faible d'adressage de l'EPROM

L'adressage de poids fort de l'EPROM consistant à appliquer R (1).1 sur les lignes A_8 à A_{10} pose aussi un problème car si A_8 et A_9 sont bien reliées dans l'ordre à PA_0 et PA_1, A_{10} est reliée à PA_3 laissant PA_2 pour la commande $\overline{OE}$. De plus, PA_5 est relié à $\overline{CE}$/PGM. On ne peut donc pas simplement tranférer R (1).1 sur M (2000). Le poids fort de l'adresse R (1).1, soit $00000A_{10}A_9A_8$, est d'abord chargé sur D. L'addition de 00000100, soit 04 en hexadécimal, a pour but de décaler A_{10} vers la gauche : si $A_{10} = 1$, l'addition avec 1 crée une retenue qui se place en D_3 ; si $A_{10} = 0$, pas de retenue et D_3 reste à 0. Le *« ou immédiat »* effectué avec 00000100 met D_2 à 1, s'il ne l'était pas encore. A l'issue de ces opérations, D contient

$0000A_{10}1A_9A_8$. Le stockage sur le port A assure la réalisation de l'adressage et la mise de $\overline{OE}$ à 1 et de $\overline{CE}$/PGM à 0.

Ces préparatifs effectués, la mise à 1 de $\overline{CE}$/PGM est réalisée en ajoutant 00100000 à D et en stockant le résultat sur le port A (ceci ne modifie ni l'adresse ni $\overline{OE}$ qui reste à 1).

La temporisation de 50 ms fait naturellement appel au sous-programme du moniteur. Attention de bien utiliser une horloge de période 2 MHz car la notice de l'EPROM 2716 indique qu'il ne faut pas dépasser la durée de programmation de 55 ms.

La remise de $\overline{CE}$/PGM à 0 est faite par l'opération inverse, soustraction de 00100000 du contenu initial de D, sauvegardé transitoirement dans R (E).0 car le sous-programme de temporisation utilise abondamment l'accumulateur.

La programmation d'une ligne effectuée, les registres d'adressage de la mémoire du système (R (A)) et de l'EPROM (R (1)) sont incrémentés pour préparer la programmation éventuelle de la ligne suivante et le registre compteur d'octets (R (B)) est décrémenté. L'opération recommence ainsi tant que la liste d'octets n'est pas épuisée.

Notez que le branchement issu du test de R (B) se fait à l'adresse 024D non pas pour reprendre la programmation en sortie des ports A, B et C mais pour assurer la réinitialisation de X et R (7).

Tel qu'il est fourni, ce programme résulte de la simplification d'un programme plus ambitieux prévoyant notamment une procédure de vérification à l'issue de la programmation de chaque ligne. Cette vérification s'est avérée inutile car la programmation s'effectuait toujours correctement. Il en découle que ce programme pourrait être repensé et restructuré et probablement réduit.

III. Lecture ou vérification de virginité d'une EPROM 2716

III. 1. Le but et les méthodes

Le but initial que nous somme fixé est la vérification de virginité d'une zone d'EPROM avant d'en effectuer la programmation. Cette opération est sans doute superflue si l'on travaille avec une EPROM qui n'a jamais servi. C'est une mesure de prudence qui s'impose lorsque l'on utilise une EPROM qui a déjà été programmée et effacée : il faut vérifier que l'exposition au rayonnement UV a été suffisante pour annihiler toute trace de programmation (l'EPROM ne contient alors que des 1). C'est encore une nécessité quand on programme une EPROM « par morceaux » et que l'on n'a pas pris toujours bien soin de noter quelles sont les lignes déjà utilisées.

Naturellement, préalablement au lancement d'un programme de vérification de virginité, il faut fournir au système les données dont il a besoin : adresse du début de la zone EPROM à vérifier écrite aux lignes 0C04 et 0C05 de la RAM 128 octets et nombre de lignes à vérifier inscrit aux lignes

0C02 et 0C03. Ces mêmes données sont réutilisées pour la programmation de la zone EPROM.

Quand le test d'une ligne conduit à un résultat négatif (cette ligne ne contient pas que des 1), il faut que l'expérimentateur soit prévenu. Le procédé qui vient immédiatement à l'esprit consiste à envisager l'affichage de l'adresse de cette ligne pendant un temps suffisamment long pour que l'on puisse la noter, avant d'effectuer le test de la ligne suivante.

Le test de chaque ligne de la zone à vérifier nécessite bien évidemment une lecture de cette ligne pour comparer son contenu à FF. Comme il faut en plus mettre en œuvre la procédure d'affichage de l'adresse d'une ligne non vierge, pourquoi ne pas en profiter pour afficher en même temps la donnée lue ?

C'est exactement ce que nous avons décidé de réaliser. Quand le contenu d'une ligne testée est FF (ligne vierge), on n'affiche rien. Quand le contenu est différent de FF, on affiche adresse et contenu. De cette manière, la vérification de virginité d'une zone EPROM se transforme en une véritable lecture (avec affichage d'adresse et de donnée) de la zone EPROM.

Remarque

On aurait pu raisonner en sens inverse et prévoir d'abord la lecture (avec affichage) de la zone EPROM. Dans le cas où le résultat affiché de la lecture de chaque ligne est toujours FF, on en déduit que la zone EPROM est vierge. Remarquez que cette procédure appliquée à la vérification de virginité est longue à cause de l'affichage. Si l'on consacre 2 s à l'affichage de l'adresse et du contenu de chaque ligne, la vérification de virginité de l'EPROM complète nécessite $2\,048 \times 2 = 4\,096$ s soit, plus d'une heure, alors qu'avec la solution adoptée (pas d'affichage si la ligne est vierge) il suffit d'environ 2 s.

N'oublions pas que, en conséquence du mode choisi, l'absence d'affichage pour une ligne quand on procède à la lecture d'une EPROM doit être traduit par : cette ligne contient FF.

Les notices de l'EPROM indiquent que pour effectuer la lecture d'une ligne, il faut imposer $\overline{OE} = 0$ et $\overline{CE}/PGM = 0$ et bien entendu placer sur les lignes A_0 à A_{10} le code d'adressage de cette ligne. La broche V_{pp} est normalement mise au potentiel 5 V (position « non » de l'inverseur). Si V_{pp} est resté à 25 V, cela ne porte pas à conséquence fâcheuse.

III. 2. Programmation de la lecture ou de la vérification de virginité

On consultera la figure IX. 3 et le tableau IX. 3.

Le registre R (1) destiné à fournir l'adresse de la ligne EPROM à lire et le registre R (B) contenant le nombre de lignes restant à tester sont d'abord chargés respectivement par l'adresse de la première ligne EPROM et le nombre total des lignes à tester de deux manières différentes :

— ces données ont été écrites préalablement en 0C04-0C05 et en

Adresses	Codes et opérandes	Mnémoniques	Commentaires
Initialisations pour lire ou vérifier la virginité de toute une EPROM 2716			
0200	F8 00	LDI 00	
02	B1	PHI R (1)	
03	A1	PLO R (1)	R (1) contient 0000
04	AB	PLO R (B)	
05	F8 08	LDI 08	
07	BB	PHI R (B)	R (B) contient 0800
08	30 0D	BR 0D	Branchement ou programme de lecture proprement dit
Initialisations pour lire ou vérifier la virginité d'une zone d'EPROM 2716			
020A	D4 01CF	SEP R (4)	Appel du S.P. de chargement du registre d'adresse EPROM et du compteur d'octets
Programme de lecture ou de vérification de virginité			
020D	F8 02	LDI 02	
0F	A7	PLO R (7)	R (7) pointe le port C
10	D4 0303	SEP R (4)	Appel du S.P. de formation poids faible adressage
13	90	GHI R (0)	
14	57	STR R (7)	Adressage poids faible réalisé
15	27	DEC R (7)	
16	27	DEC R (7)	R (7) pointe le port A
17	91	GHI R (1)	
18	FC 04	ADI 04	
1A	F9 04	ORI 04	
1C	FF 04	SMI 04	
1E	57	STR R (7)	Adressage poids fort réalisé. $\overline{OE}$ = 0 - $\overline{CE}$ = 0
1F	17	INC R (7)	R (7) pointe le port B
20	07	LDN R (7)	Donnée EPROM chargée et
21	AD	PLO R (D)	transférée dans R (D). 0
22	E7	SEX R (7)	Donnée comparée à FF D mis à 0 si donnée = FF
23	F8 FF	LDI FF	
25	F5	SD	

Adresses	Codes et opérandes	Mnémoniques	Commentaires
26	17	INC R (7)	R (7) réinitialisé pour pointer le port C
27	3A 34	BNZ 34	Branchement à affichage adresse et donnée si donnée = FF
29	11	INC R (1)	
2A	2B	DEC R (B)	
2B	9B	GHI R (B)	Test de R (B) et branchement pour lire ou vérifier la ligne suivante si zone non épuisée
2C	3A 0D	BNZ 0D	
2E	8B	GLO R (B)	
2F	3A 0D	BNZ 0D	
31	C0 0037	LBR	
Affichage de l'adresse et de la donnée si la donnée diffère de FF			
0234	91	GHI R (1)	Adresse EPROM transférée dans R (A) en vue d'affichage.
35	BA	PHI R (A)	
36	81	GLO R (1)	
37	AA	PLO R (A)	
38	D4 0193	SEP R (4)	Appel du S.P. de stockage des codes d'affichage de l'adresse
3B	D4 01CC	SEP R (4)	Appel du S.P. de stockage des codes d'affichage de la donnée
3E	F8 FF	LDI FF	Initialisation d'un compteur de boucles à FF
40	AE	PLO R (E)	
41	D4 010D	SEP R (4)	Appel du S.P. d'affichage
44	2E	DEC R (E)	
45	8E	GLO R (E)	Test de R(E) et branchement pour effectuer 256 fois l'affichage
46	3A 41	BNZ 41	
48	30 29	BR 29	Retour à la branche principale du programme

Tableau IX. 3. Programme de lecture ou de vérification de virginité d'une EPROM

0C02-0C03 grâce au moniteur et elles sont transférées dans R (1) et R (B) grâce au sous-programme spécifique d'adresse 01CF appelé à l'adresse 020A ;

— ces données sont respectivement 0000 et 0800 lorsque l'on veut lire ou vérifier la virginité d'une EPROM 2716 entière ; un programme élémen-

taire écrit à partir de l'adresse 0200 les charge respectivement dans R (1) et R (B).

L'adressage de poids faible s'effectue — comme pour la programmation — en ressortant sur le port C le registre R (1).0 après mise en ordre convenable des bits grâce au sous-programme d'adresse 0303.

L'adressage de poids fort (voir programme de programmation) réalise en plus $\overline{CE} = 0$ et $\overline{OE} = 0$ par une soustraction supplémentaire de 00000100.

L'EPROM en fonction lecture et correctement adressée présente alors sa donnée au port B programmé en entrée lors de l'initialisation du moniteur. Cette donnée est lue sur le port B et transférée sur le registre R (D).0 en vue de son affichage éventuel (R (D).0 est le registre d'affichage de donnée du moniteur).

De la donnée toujours présente sur le port B, on soustrait ensuite FF contenu dans D. Si le résultat est différent de 0, la donnée n'est pas FF et il faut afficher. L'adresse EPROM contenue dans R (1) est alors transférée dans R (A), registre d'affichage d'adresse du moniteur. Le moniteur contient les sous-programmes qui permettent de former et stocker en mémoire RAM les codes d'affichage de l'adresse contenue dans R (A) et de la donnée contenue dans R (D).0. L'affichage de l'adresse et de la donnée EPROM est réalisé et maintenu en faisant appel 256 fois de suite au sous-programme d'affichage (réalisation simultanée du rafraîchissement et du réglage de la durée d'affichage par la boucle de compteur R (E).0.

Si le résultat de la soustraction est 0, c'est-à-dire si la donnée est FF, l'affichage ne se fait pas. Directement R (1) est incrémenté pour préparer la lecture de la ligne suivante et R (B) est décrémenté. Tant qu'il reste des lignes à lire, un branchement s'effectue pour reprendre adressage et lecture. Sinon un branchement s'effectue au moniteur qui s'initialise pour indiquer PrEt.

IV. Procédure d'emploi

On suppose que les maquettes VILÉMIO 1, 2 et 3 sont reliées. Pour effectuer une opération de programmation ou de lecture d'une EPROM 2716, il est *conseillé* de placer celle-ci sur le support à force d'insertion nulle de la maquette 2 *avant* de brancher l'alimentation 5 V de l'ensemble. L'alimentation branchée, l'afficheur indique PrEt. Le commutateur d'alimentation de la broche V_{pp} est en position « non ».

IV. 1. Contrôle de virginité ou lecture d'une EPROM 2716 entière

— Frapper l'adresse 0200 du programme correspondant.

— Appuyer sur la touche P pour lancer le programme.

Si l'EPROM est entièrement vierge, l'afficheur indique à nouveau PrEt et le moniteur peut être utilisé à une autre tâche. Si ce n'est pas le cas,

l'afficheur indique successivement les adresses et contenus *des lignes non vierges ;* quand toutes les lignes ont été testées, le contrôle est rendu au moniteur et il s'affiche PrEt.

IV. 2. Contrôle de virginité ou lecture d'une zone d'EPROM 2716

— En s'aidant du moniteur, écrire aux adresses 0C02 et 0C03 respectivement le poids fort N.1 et le poids faible N.0 du nombre N des lignes à vérifier ou lire.

Remarque

Bien entendu ce nombre N doit être exprimé dans le système hexadécimal. Dans le cas où le calcul de N en hexadécimal serait jugé ennuyeux ou fastidieux, on pourrait utiliser la procédure de détermination automatique exposée au chapitre 11.

— Ecrire aux adresses 0C04 et 0C05 respectivement le poids fort et le poids faible de l'adresse de la première ligne de la zone EPROM à vérifier ou lire.

— Après avoir réinitialisé le moniteur par action sur la touche I, frapper l'adresse 0204 du programme de vérification de virginité ou lecture d'une zone EPROM.

— Appuyer sur la touche P pour lancer le programme. Quand l'opération au cours de laquelle les contenus *des lignes non vierges* et leurs adresses s'affichent successivement est terminée, l'afficheur indique PrEt.

IV. 3. Programmation d'une zone EPROM

— En s'aidant du moniteur, écrire successivement :

— aux adresses 0C00 et 0C01 respectivement le poids fort et le poids faible de l'adresse de la ligne à partir de laquelle est écrite en mémoire adressée directement par le microprocesseur le programme à inscrire dans la zone EPROM (on l'appelle adresse de la zone source) ;

— aux adresses 0C02 et 0C03 respectivement le poids fort et le poids faible du nombre de lignes à programmer (nombre d'octets à transférer en EPROM). Même remarque qu'en IV. 2.

— aux adresses 0C04 et 0C05 respectivement le poids fort et le poids faible de l'adresse de la première ligne de la zone EPROM à programmer (adresse de la zone destination).

— Après avoir réinitialisé le moniteur par action sur la touche I, frapper l'adresse 024A du programme de programmation d'une zone EPROM.

— Si ce n'est pas déjà fait, brancher l'alimentation stabilisée réglée à 25 V destinée à V_{pp} et mettre l'inverseur correspondant en position « oui ».

— Appuyer sur la touche P pour lancer le programme. Quand la programmation est terminée, l'afficheur indique PrEt.

— Remettre l'inverseur d'alimentation V_{pp} en position « non ».

Nota : La programmation d'une ligne demande un peu plus de 50 ms. Pour programmer une EPROM 2716 en entier (2 048 lignes) il faut donc près de 2 mn.

Chapitre 10

ENREGISTREMENT ET LECTURE SUR CASSETTE D'UNE SÉRIE D'OCTETS

Le problème de la conservation d'un programme ou d'une table de données et de leur restitution à une RAM (ou recharge) peut être résolu assez simplement par l'utilisation d'un magnétophone à cassettes de faible coût. Ce magnétophone doit comporter une *entrée* d'enregistrement (dont l'utilisation met hors service le microphone) et une *sortie* destinée à l'écoute par l'intermédiaire d'un casque, d'un petit écouteur d'oreille ou d'un haut-parleur auxiliaire (l'utilisation de cette sortie mettant généralement hors service le haut-parleur intégré au magnétophone).

On peut ainsi se constituer une « bibliothèque » de programmes et une « banque » de données.

La liaison entre le système microprocesseur-mémoire-organes d'entrée/sortie et le magnétophone est assurée par un circuit spécifique appelé *interface cassette,* prévu sur la maquette « mise au point » de l'ensemble VILÉMIO. Bien entendu, c'est encore le microprocesseur associé à sa mémoire de programme qui est chargé de gérer les opérations de transfert de la mémoire vers la cassette ou de la cassette vers la mémoire. La figure X. 1 rappelle la structure du dispositif.

I. Principe utilisé pour l'enregistrement et la lecture d'un bit

I. 1. Chaque bit à *enregistrer* donne lieu à la création d'un train d'impulsions suivi d'une pause.

L'intervalle de temps attribué à la transmission d'un bit est partagé en trois parties (fig. X. 2) :

— Le premier tiers est toujours occupé par un train d'impulsions issues d'un oscillateur astable. Le train d'impulsions est destiné à « marquer » le début de la transmission d'un bit.

— Le deuxième tiers est occupé par un train d'impulsions si le bit à transmettre est 1, par une pause si le bit à transmettre est 0.

— Le troisième tiers est toujours une pause. Dans les deux cas elle correspond à la fin de transmission d'un bit.

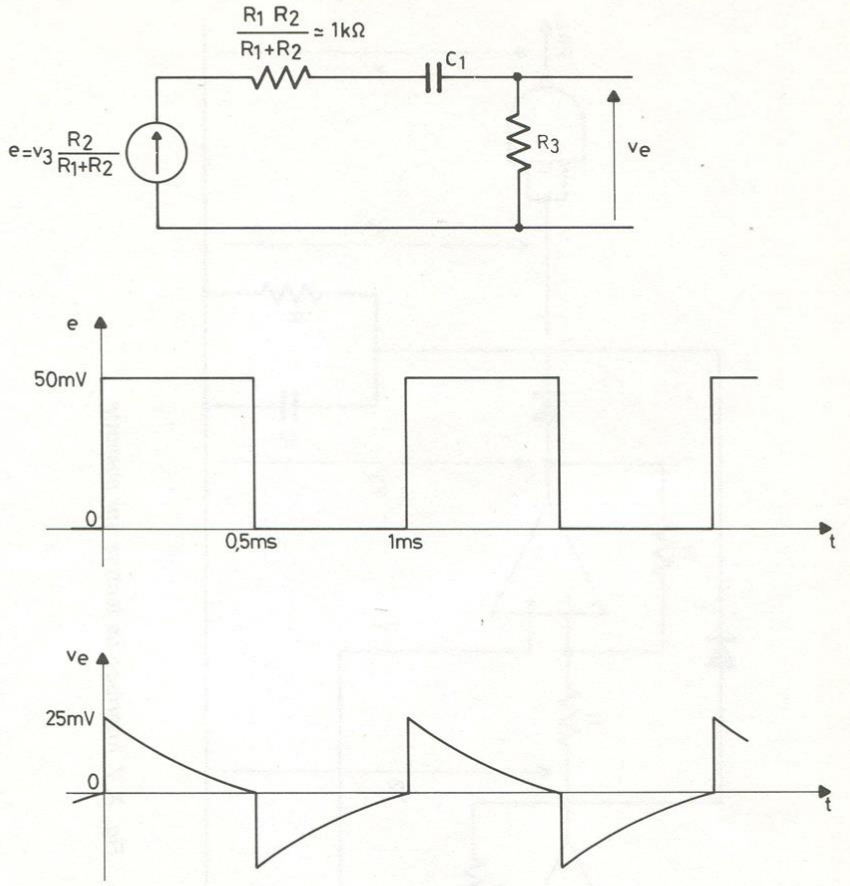


Fig. X. 6. Réponse de l'adaptateur-dérivateur

Nous supposons qu'une source auxiliaire de tension délivre la référence V = 2 V.

Examinons le fonctionnement de ce montage (suivre les formes des signaux sur la figure X. 8). Le magnétophone restitue sur sa sortie pour haut-parleur auxiliaire ou casque une tension v_s qui, à cause de la limitation de la bande passante de l'appareil, est plus « arrondie » que la tension v_e reçue. On suppose que, grâce au réglage du volume, la valeur de crête du signal de sortie v_s est 0,5 V.

III. 1. La cellule C, R_o avec C = 0,1μF et R_o de l'ordre du mégohm n'affecte

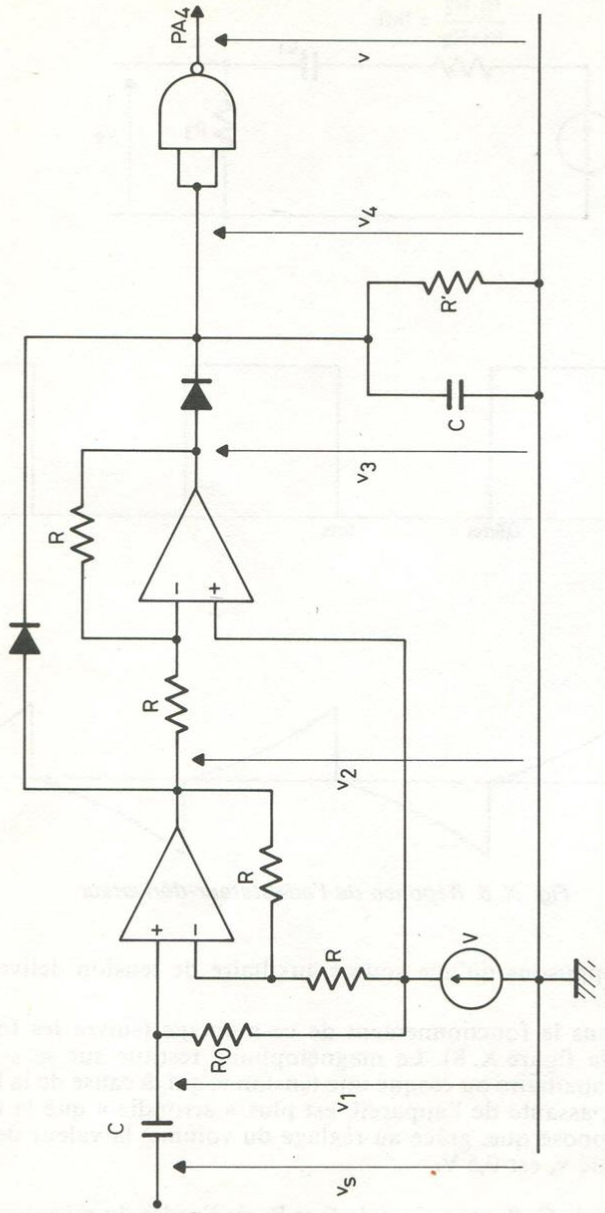


Fig. X. 7. Interface de lecture sur cassette

pratiquement pas le signal reçu. v_1 recopie v_s à la composante continue $V = 2$ V près, soit $v_1 = v_s + V$. La résistance R_o est nécessaire pour l'écoulement du courant de polarisation (très faible pour un BIFET mais existant malgré tout).

III. 2. Le montage non inverseur : $R = 1 k\Omega$

Les deux résistances R étant identiques, le potentiel par rapport à la

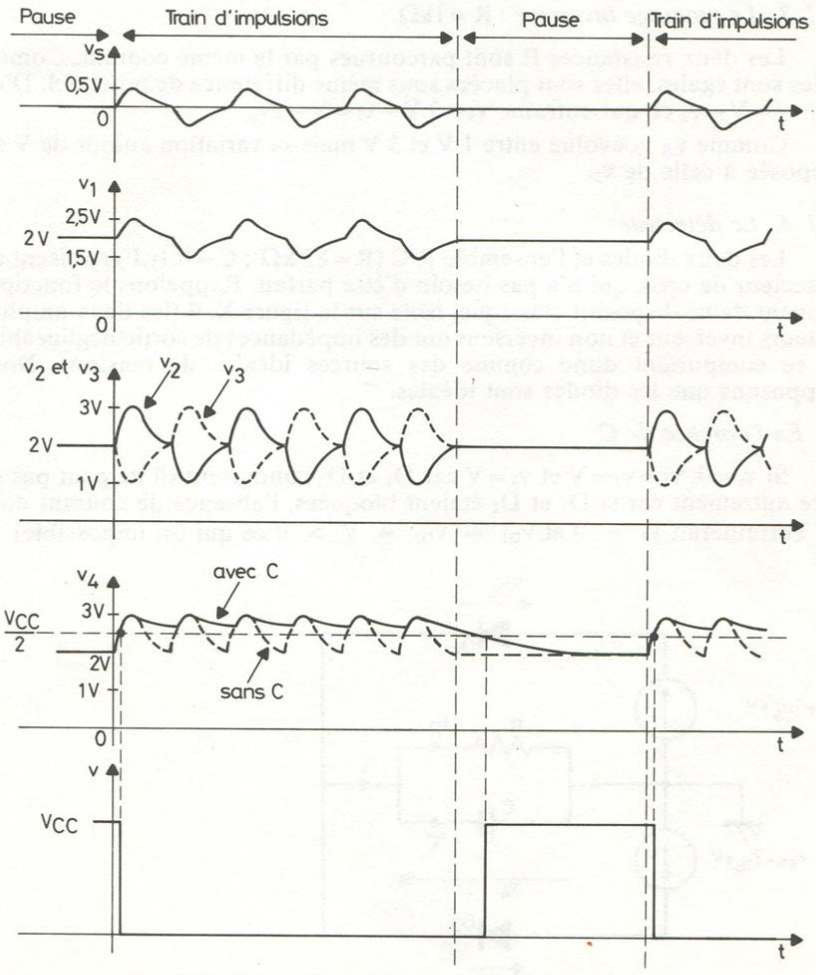


Fig. X. 8. Forme des signaux de l'interface de lecture

masse de l'entrée inverseuse est $\frac{v_2+V}{2}$. Ce potentiel est le même que celui de l'entrée non inverseuse soit :

$$v_1 = v_s+V = \frac{v_2+V}{2} \quad \text{ce qui entraîne} \quad v_2 = 2v_s+V.$$

Avec notre hypothèse, v_2 évolue entre 1 V et 3 V.

III. 3. Le montage inverseur : $R = 1k\Omega$.

Les deux résistances R sont parcourues par le même courant. Comme elles sont égales, elles sont placées sous même différence de potentiel. D'où $v_3 - V = V - v_2$ ce qui entraîne $v_3 = 2V - v_2 = V - 2v_s$.

Comme v_2, v_3 évolue entre 1 V et 3 V mais sa variation autour de V est opposée à celle de v_2.

III. 4. Le détecteur

Les deux diodes et l'ensemble R'C ($R = 82\ k\Omega$; $C = 0{,}1\mu F$) réalisent un détecteur de crête qui n'a pas besoin d'être parfait. Rappelons le fonctionnement de ce dispositif classique, isolé sur la figure X. 9 (les deux amplificateurs inverseur et non inverseur ont des impédances de sortie négligeables et se comportent donc comme des sources idéales de tension). Nous supposons que les diodes sont idéales.

a) *En l'absence de C*

Si $v_s = 0$, $v_3 = v_2 = V$ et $v_4 = V$ car D_1 et D_2 conduisent (il ne peut pas en être autrement car si D_1 et D_2 étaient bloquées, l'absence de courant dans R' entraînerait $v_4 = 0$ et $v_{D1} = v_{D2} = V > 0$ ce qui est impossible)

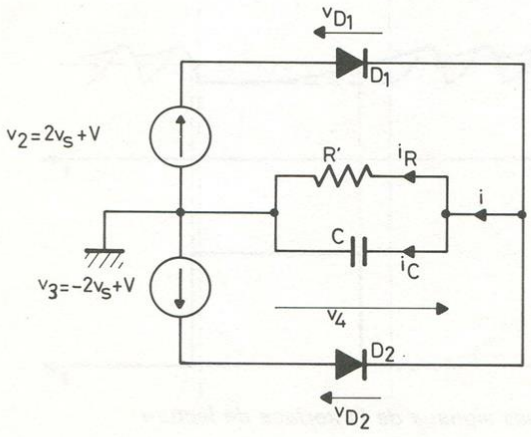


Fig. X. 9. Détecteur

Si $v_s > 0$, D_1 conduit, $v_{D1} = 0$ et $v_{D2} = -4v_s < 0$. D_2 est donc bloquée et $v_4 = v_2$.

Si $v_s < 0$, c'est D_2 qui conduit et c'est D_1 qui est bloquée et $v_4 = v_3$. v_4 a alors l'allure indiquée en pointillés sur la figure X.8.

b) *En présence de C*

C est sous différence de potentiel v_4 comme R'. La charge de C (augmentation de v_4) se fait à travers la diode qui conduit, mais la décharge (diminution de v_4) pose des problèmes et peut ne s'effectuer qu'à travers la résistance R'.

Si v_4 croît, $i_C = C\frac{dv_4}{dt} > 0$; comme $i_R = \frac{v_4}{R} > 0$, $i = i_R + i_C > 0$.

i passe par la diode qui conduit et $v_4 = v_2$ ou v_3.

Si v_4 décroît, $i_c < 0$; i_r demeure positif car v_4 est toujours positif. Tant que $i = i_r + i_c > 0$, une diode conduit et $v_4 = v_2$ ou v_3. Quand v_4 est suffisamment faible pour que i s'annule, la diode qui conduisait se bloque aussi. C continue à se décharger dans R' et v_4 évolue alors exponentionellement suivant la loi $v_4 = K \exp\left(-\frac{t}{R'C}\right)$ jusqu'à ce que la conduction d'une diode reprenne.

v_4 a l'allure indiquée en traits pleins sur la figure X.8.

III. 5. Le comparateur

Sa fonction est très simple : tant que $v_4 > \frac{V_{cc}}{2}$, seuil de basculement, $v = 0$; si $v_4 < \frac{V_{cc}}{2}$, $v = +V_{cc}$.

On obtient ainsi la tension v que l'on appliquera à la broche PA_4 du 8255. Notons que le niveau logique de v est 0 sur une durée légèrement supérieure à celle de la durée du train d'impulsions initial. Ceci n'a aucune incidence sur le bon fonctionnement du système. En effet, la seule condition à remplir est que, sur l'intervalle de temps consacré à la transmission d'un bit, v soit au niveau 0 *plus longtemps* qu'au niveau 1 si ce bit est 1, *moins longtemps* si ce bit est 0.

III. 6. Remarque :

Matériellement, la source de référence V est créée grâce à un diviseur de tension utilisant deux résistances de 330 Ω. Partant de la tension d'alimentation de 5 V, on obtient ainsi un dipôle de f-e-m 2,5 V et de résistance 115 Ω. Cette résistance ne perturbe pratiquement pas le fonctionnement du montage. Les 2,5 V diffèrent sensiblement de la valeur 2 V utilisée dans l'explication ci-dessus. La différence est à peu près compensée par le seuil de conduction des deux diodes D_1 et D_2. En définitive, la forme du signal v_4 est bien celle indiquée sur la figure X.8.

IV. Programmation de l'enregistrement

IV. 1. Intéressons-nous d'abord à l'organigramme de la *phase d'enregistrement d'un bit* (fig. X. 10)

La bascule Q est d'abord mise à 0 de manière à commander la mise en marche de l'oscillateur. On laisse osciller pendant par exemple 30 ms. Le bit à enregistrer, qui est le bit de plus faible poids de R (6).0, est chargé dans DF. DF est testé : si le bit est 0, Q est mis à 1 pour arrêter l'oscillateur ; si le bit est 1, l'oscillateur continue à fonctionner car Q est laissé à 0. On reste dans cette situation pendant 30 ms au bout desquelles l'oscillateur est bloqué pendant 30 nouvelles ms par la mise de Q à 1 (si ce n'était pas déjà fait).

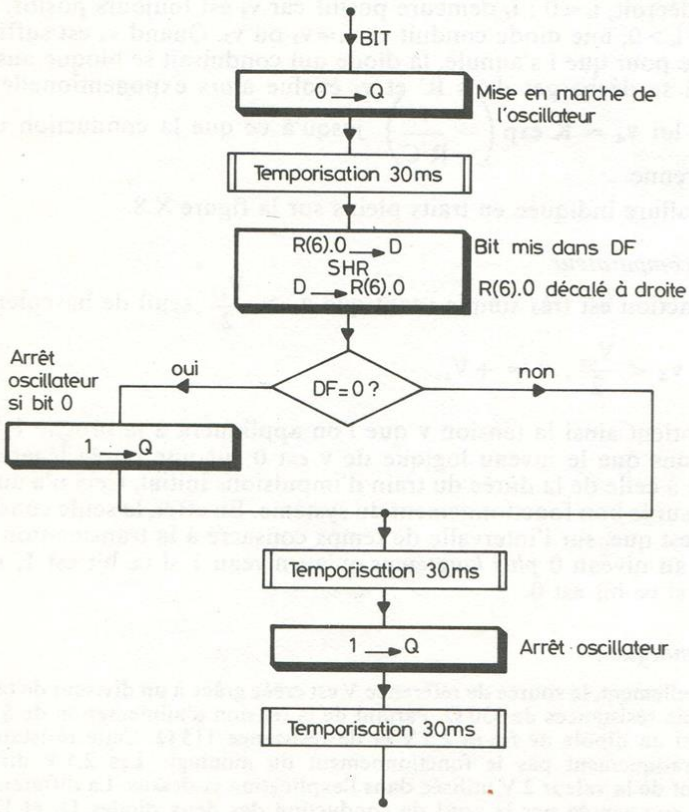


Fig. X. 10. Organigramme de la phase d'enregistrement d'un bit

Le programme correspondant peut être lu à partir de l'adresse BIT= 0289 (tableau X.1).

Étiquettes	Adresses	Codes et opérandes	Mnémoniques	Commentaires
	027B	D4 01CF	SEP R (4)	Appel du S.P. de chargement du registre d'adresse et du compteur d'octets
	7E	7B	SEQ	Arrêt oscillateur
	7F	D4 0100 04D2	SEP R (4)	Appel du S.P. de temporisation... ... de 30 ms
OCT	84	F8 09	LDI 09	Compteur de bits R (9). 0 chargé à 9
	86	A9	PLO R (9)	
	87	4A	LDA R (A)	Octet à enregistrer dans R (D). 0 R (A) pointe la ligne suivante
	88	AD	PLO R (D)	
BIT	89	7A	REQ	Marche oscillateur
	8A	D4 0100 04D2	SEP R (4)	Appel du S.P. de temporisation ... de 30 ms
	8F	8D	GLO R (D)	Bit à enregistrer mis dans DF R (D). 0 décalé à droite
	90	F6	SHR	
	91	AD	PLO R (D)	
	92	3B 95	BNZ	Branchement pour arrêt oscillateur si bit = 0
	94	38	SKP	Saut d'une ligne
	95	7B	SEQ	Arrêt oscillateur si bit = 0
	96	D4 0100 04D2	SEP R (4)	Appel du S.P. de temporisation... ... de 30 ms
	9B	7B	SEQ	Arrêt oscillateur si bit = 0
	9C	D4 0100 04D2	SEP R (4)	Appel du S.P. de temporisation... ... 30 ms
	A1	29	DEC R (9)	
	A2	89	GLO R (9)	
	A3	3A 89	BNZ BIT	Branchement à BIT si octet non terminé
	A5	2B	DEC R (B)	
	A6	9B	GHI R (B)	Branchement à OCT si liste d'octets non épuisée
	A7	3A 84	BNZ OCT	
	A9	8B	GLO R (B)	
	AA	3A 84	BNZ OCT	
	AC	7A	REQ	Remise en marche de l'oscillateur
	AD	C0 0037	LBR	Retour au moniteur

Tableau X. 1 Programme d'enregistrement sur cassette d'une série d'octets

IV. 2. L'enregistrement d'un octet consiste à enregistrer consécutivement huit bits placés initialement dans un registre.

Examinons l'organigramme de la figure X. 11 à partir du repère OCT. Le registre R (9).0, baptisé compteur de bits, est chargé à 9. Ceci peut paraître surprenant puisque l'on désire transmettre 8 bits ! En fait l'octet à enregistrer est puisé dans la mémoire et placé sur le registre de travail R (D).0. La phase d'enregistrement d'un bit (fig. X.10) utilise le bit de droite de R (6).0 et laisse ce registre décalé vers la droite pour préparer la transmission du bit suivant, un 0 ayant remplacé le bit initial de plus fort poids. Quand les huit bits de l'octet sont enregistrés, R (6).0 ne contient plus que des 0. Le bit de plus faible poids (c'est-à-dire 0 comme les autres) est alors enregistré. Ce neuvième bit marque la fin de l'enregistrement d'un octet et est utile pour l'opération de relecture de la bande magnétique.

IV. 3. L'enregistrement d'une série de N octets nécessite de faire N fois cette opération. L'adresse du premier octet à enregistrer a été écrite grâce au moniteur aux lignes 0C00 et 0C01 alors que le nombre N a été placé aux lignes 0C02 et 0C03. Un sous-programme spécifique charge ces deux données respectivement dans les registres R (A) (pointeur de la zone mémoire où se situent les octets à transférer) et R (B) (compteur d'octets qui contiendra toujours le nombre des octets restant à enregistrer).

Bien entendu, le magnétophone doit être mis en marche (sur la fonction enregistrement) avant le lancement du programme d'enregistrement. Comme la bascule Q est normalement à l'état 0, l'oscillateur fonctionne et une inscription est faite sur la cassette. De manière à distinguer cette inscription de celle que laisse le début de l'enregistrement du premier bit, il est nécessaire de prévoir un « blanc » entre les deux. Aussi l'oscillateur est-il bloqué pendant par exemple 30 ms par la mise de Q à 1. Il faut comprendre que cette opération est indispensable pour que, à la lecture de la bande, puisse être détecté le premier front descendant de v correspondant au début de la transmission du premier bit.

De la même manière, avant le retour au moniteur, la bascule Q est remise à 0 (oscillateur remis en marche) pour marquer la fin de l'enregistrement du dernier bit (fin de la pause).

Le programme complet d'enregistrement apparaît sur le tableau X. 1.

V. Programmation de la lecture

V. 1. Commençons par comprendre comment est effectuée *la lecture d'un bit* et pour cela examinons l'organigramme de la figure X. 12 à partir du repère BIT.

Faisons l'hypothèse qu'un front descendant du signal v vienne d'être détecté ; la transmission d'un bit commence donc. Le registre R (E), compteur d'état, est mis à 0. Il est ensuite décrémenté d'une unité toutes les

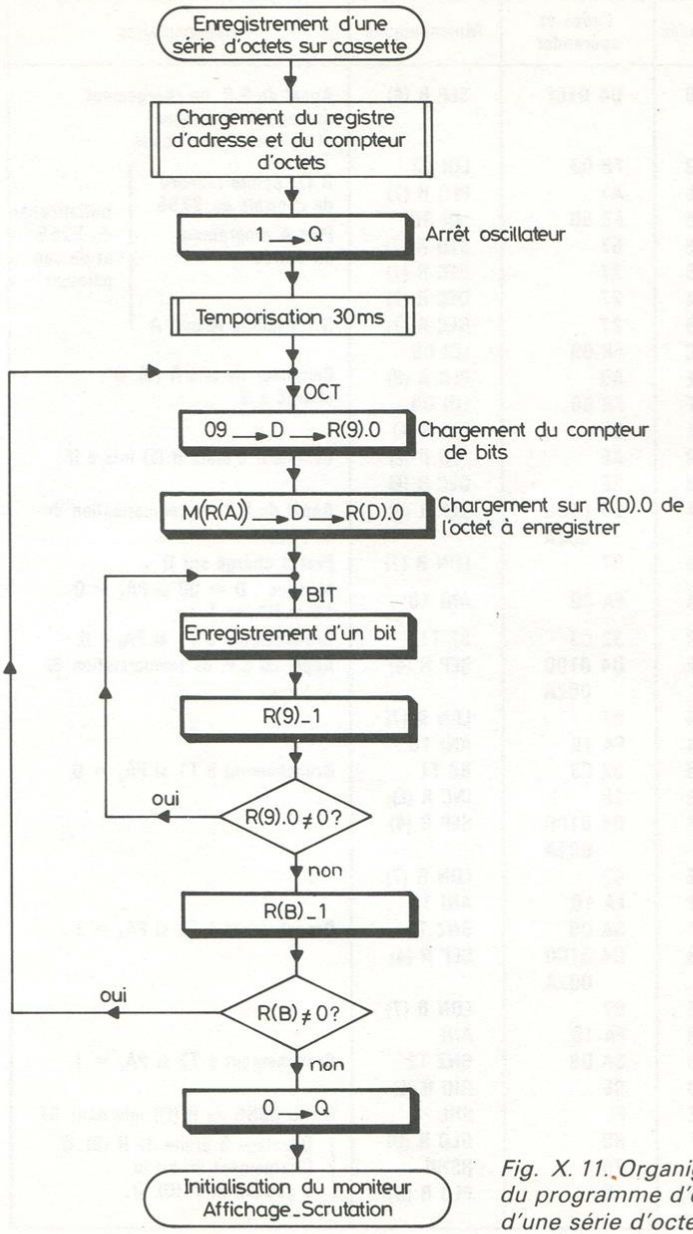


Fig. X. 11. Organigramme du programme d'enregistrement d'une série d'octets

Étiquettes	Adresses	Codes et opérandes	Mnémoniques	Commentaires
	02B0	D4 01CF	SEP R (4)	Appel du S.P. de chargement du registre d'adresse et du compteur d'octets
	B3	F8 03	LDI 03	R (7) pointe registre de contrôle du 8255 — Initialisation du 8255 et de son pointeur
	B5	A7	PLO R (7)	
	B6	F8 90	LDI 90	Port A programmé en entrée
	B8	57	STR R (7)	
	B9	27	DEC R (7)	
	BA	27	DEC R (7)	
	BB	27	DEC R (7)	R (7) pointe le port A
OCT	BC	F8 09	LDI 09	
	BE	A9	PLO R (9)	Compteur de bits R (9). 0 chargé à 9
BIT	BF	F8 00	LDI 00	
	C1	BE	PHI R (E)	
	C2	AE	PLO R (E)	Compteur d'états R (E) mis à 0.
T1	C3	2E	DEC R (E)	
	C4	D4 0100 002A	SEP R (4)	Appel du S.P. de temporisation de... ... 1 ms
	C9	07	LDN R (7)	Port A chargé sur D
	CA	FA 10	ANI 10	Masque : D = 00 si PA_4 = 0, 10 si PA_4 = 1
	CC	32 C3	BZ T1	Branchement à T1 si PA_4 = 0
	CE	D4 0100 002A	SEP R (4)	Appel du S.P. de temporisation de... ... 1 ms
	D3	07	LDN R (7)	
	D4	FA 10	ANI 10	
	D6	32 C3	BZ T1	Branchement à T1 si PA_4 = 0.
T2	D8	1E	INC R (E)	
	D9	D4 0100 002A	SEP R (4)	
	DE	07	LDN R (7)	
	DF	FA 10	ANI 10	
	E1	3A D8	BNZ T2	Branchement à T2 si PA_4 = 1
	E3	D4 0100 002A	SEP R (4)	
	E8	07	LDN R (7)	
	E9	FA 10	ANI	
	EB	3A D8	BNZ T2	Branchement à T2 si PA_4 = 1
	ED	9E	GHI R (E)	
	EE	FE	SHL	Bit lu (MSB de R (E)) mis dans DF
	EF	8D	GLO R (D)	Décalage à droite de R (D). 0 Chargement du bit lu à gauche de R (D). 0
	F0	76	RSHR	
	F1	AD	PLO R (D)	

Étiquettes	Adresses	Codes et opérandes	Mnémoniques	Commentaires
	F2	29	DEC R (9)	
	F3	89	GLO R (9)	
	F4	3A BF	BNZ BIT	Branchement à BIT si octet non
	F6	8D	GLO R (D)	terminé
	F7	5A	STR R (A)	Stockage en RAM de l'octet lu
	F8	1A	INC R (A)	R (A) pointe la ligne suivante
	F9	2B	DEC R (B)	
	FA	9B	GHI R (B)	Branchement à OCT si liste
	FB	3A BC	BNZ OCT	d'octets non épuisée
	FD	8B	GLO R (B)	
	FE	3A BC	BNZ OCT	
	0300	C0 0037	LBR	Retour au moniteur

Tableau X. 2 Programme de lecture sur cassette d'une série d'octets

millisecondes tant que PA_4, c'est-à-dire v, tension de sortie de l'interface de lecture, reste a 0 (suivre la boucle).

On a remarqué que le test de PA_4 est répété. L'intérêt de cette opération est le suivant : supposons que le système détecte $PA_4 = 1$ au moment du test : ceci se produit bien entendu si v est passé normalement au niveau logique 1 mais cela *peut* aussi se produire accidentellement si un parasite quelconque vient malencontreusement, juste au moment du test, placer PA_4 à 1. Ce parasite fugitif a dans ce cas le même effet que la transmission d'un bit et il est interprété comment tel par le système automatique. Le résultat de la lecture est décalé d'un bit ce qui est tout à fait désastreux : la lecture n'est plus exploitable. Le deuxième test apporte donc une vérification. Comme il intervient 1 ms après le premier ; il constitue une sorte de filtre logiciel qui élimine l'effet des parasites pas trop lents. L'élimination par ce biais de l'effet d'un parasite entraîne la perte d'une décrémentation de R (E). Ce n'est pas gênant car cela ne modifiera pas le *signe* du contenu final de R (E).

Une fois détecté de façon quasi certaine le passage de v à 1, le registre R (E) est incrémenté toutes les millisecondes tant que $PA_4 = v$ est au niveau 1. La détection du passage de v à 0 est confirmée par un deuxième test de PA_4 (nouveau filtrage de parasite).

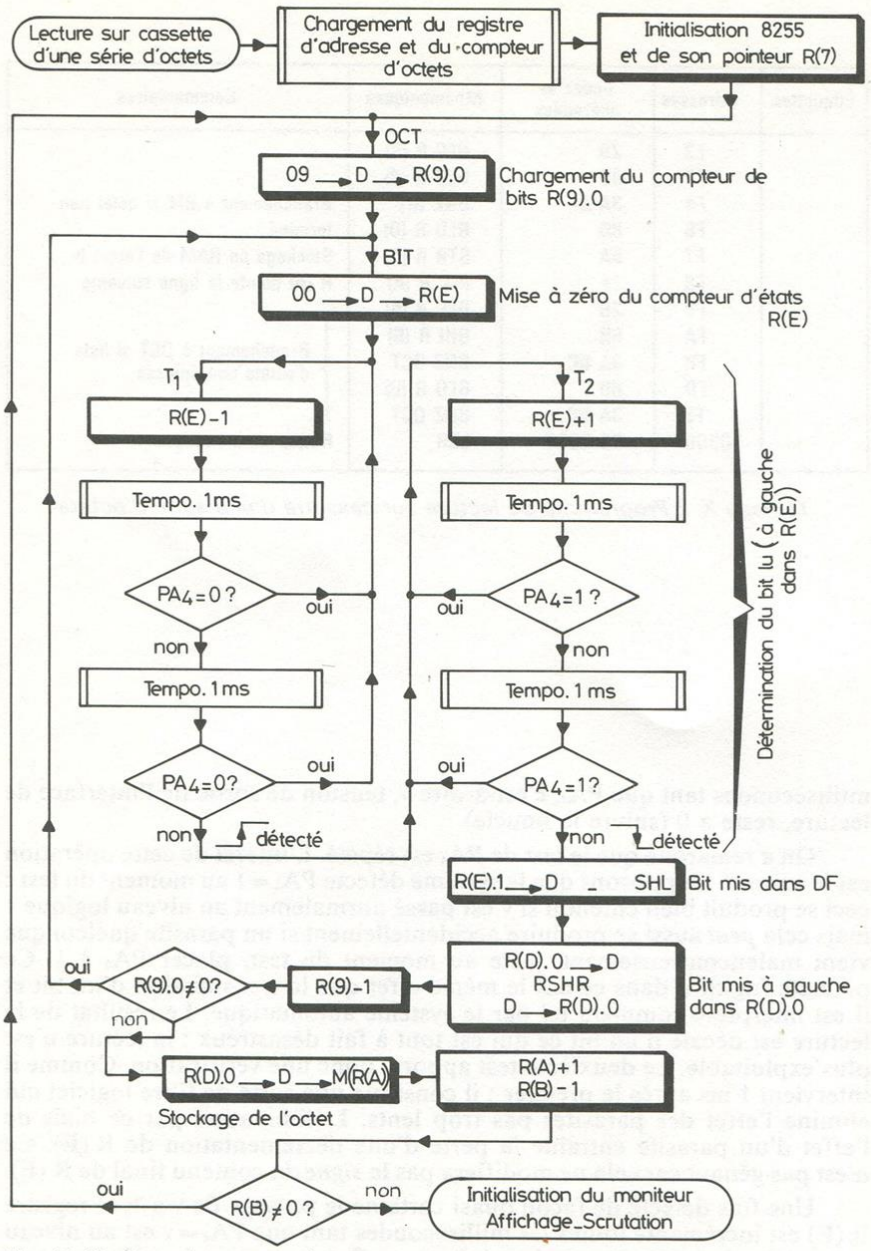


Fig. X. 12. Organigramme du programme de lecture sur cassette d'une série d'octets

Comment est réalisé pratiquement le test de PA_4 ? En début du programme de lecture, le port A du PPI 8255 est programmé en entrée (son pointeur R (7) désigne le registre de contrôle situé à l'adresse 2003 ; le code 90 y est stocké ; puis R (7) est décrémenté pour venir désigner le port A : voir le programme du tableau X. 2 et la description du 8255 au chapitre 6). Le contenu du port A est chargé sur le registre D et les bits autres que PA_4 sont masqués en opérant un « et » immédiat avec l'octet 00010000 soit 10 en hexadécimal. Le contenu de l'accumulateur est alors 0 si $PA_4 = 0$, 10 si $PA_4 = 1$. Des branchements peuvent alors être effectués si $D = 0$ (ou $PA_4 = 0$) ou bien $D \neq 0$ (ou $PA_4 = 1$).

A l'issue de la détection du front descendant de v, R (E) a été *plus* décrémenté qu'incrémenté si le bit lu est 1. Son contenu est négatif et son bit de plus fort poids est 1. R (E) a été *moins* décrémenté qu'incrémenté si le bit lu est 0. Son contenu est alors positif et son bit de plus fort poids est 0. On voit que le plus fort poids de R (E) est identique au bit lu sur la cassette. Notons que ce principe de lecture est indifférent à une légère modification de vitesse de défilement de la bande.

Le bit de plus fort poids de R (E) est ensuite chargé dans DF au moyen d'un transfert préalable dans D et d'un décalage à gauche. Il est ensuite transféré à gauche du registre R (D).0 destiné à recevoir les octets en utilisant principalement un décalage circulaire à droite.

V. 2. Pour opérer la *lecture d'un octet,* le processus précédent est effectué neuf fois grâce à l'utilisation du compteur de boucle R (9) (voir l'organigramme de la figure X. 12 à partir du repère OCT).

Pour un octet, le premier bit lu est le 0, qui sépare toujours deux octets sur la bande magnétique. A la lecture du neuvième bit, ce zéro est chassé à droite du registre R (D).0 qui contient alors effectivement les huit bits de l'octet lu. Cet octet est alors stocké en mémoire RAM à l'adresse indiqué par le registre pointeur R (A) qui est ensuite incrémenté pour préparer le stockage de l'octet suivant.

V. 3. La lecture de la série de N octets nécessite l'intervention du compteur d'octets R (B). R (A) et R (B) sont dès le début du programme chargés respectivement par l'adresse où le premier octet lu doit être stocké (cette adresse a été préalablement écrite en RAM aux lignes 0C00 et 0C01) et par le nombre N des octets à lire (écrit aux lignes 0C02 et 0C03), grâce au sous-programme spécifique.

Le programme est lancé pendant que le magnétophone restitue l'oscillation initiale enregistrée, v est alors à 0 et R (E) est décrémenté toutes les millisecondes. Quand survient le « blanc » annonçant l'arrivée du signal correspondant au premier bit enregistré, R (E) est incrémenté cycliquement. L'oscillation de début du premier bit entraîne le transfert de bit de signe de R (E) à gauche de R (D).0. C'est ce bit (aléatoirement 1 ou 0) qui sera chassé à droite lors de la neuvième description de la phase BIT du premier octet.

VI. Procédure d'enregistrement et de lecture d'une série d'octets

V. 1. Enregistrement

— Relier la sortie de l'interface d'enregistrement à *l'entrée* (d'enregistrement) du magnétophone. La liaison entre la sortie Q du microprocesseur et l'entrée de commande de l'oscillateur est déjà effectuée sur la maquette.

— Grâce au moniteur, inscrire les deux octets $\mathcal{A}_1$ et $\mathcal{A}_0$ de l'adresse du premier octet à enregistrer sur les lignes 0C00 et 0C01 ainsi que les deux octets N_1 et N_0 du nombre N d'octets à enregistrer. Si la détermination, en hexadécimal, de ce nombre N n'est pas immédiate, celui-ci pourra être calculé par la mise en œuvre d'un programme spécifique décrit au chapitre suivant.

— Toujours grâce au moniteur, écrire sur l'afficheur l'adresse 027B du programme d'enregistrement inscrit en EPROM moniteur.

— A partir d'un repère R_1 (compteur du magnétophone) qu'il convient de noter, mettre le magnétophone en marche en fonction d'enregistrement. Attendre quelques secondes — correspondant par exemple à une dizaine d'unités du compteur — pendant lesquelles la vitesse de défilement de la bande comme le fonctionnement de l'amplificateur à contrôle automatique de gain se stabilisent et l'enregistrement de l'oscillation préalable s'effectue.

— Lancer le programme d'enregistrement grâce à l'action de la touche P (le compteur du magnétophone indiquant R_2, à noter).

— Quand l'enregistrement est terminé, l'afficheur d'adresse indique PrEt. Laisser le magnétophone tourner encore quelques tours. L'arrêter et noter l'indication R_3 du compteur : la bande est utilisée entre R_1 et R_3.

Nota : l'enregistrement d'un octet requiert un peu moins de 1 s.

VI. 2. Lecture de la cassette (ou chargement de la RAM).

— Relier l'entrée de l'interface de lecture à la *sortie* casque ou haut-parleur auxiliaire du magnétophone.

— Inscrire les deux octets $\mathcal{A}_1$ et $\mathcal{A}_0$ de l'adresse $\mathcal{A}$ à partir de laquelle on désire que la série d'octets s'inscrive en RAM, sur les lignes 0C00 et 0C01, ainsi que les deux octets N_1 et N_0 du nombre N des octets à relire, sur les lignes 0C02 et 0C03.

— Préparer le lancement du programme de lecture en écrivant son adresse 02B0 sur l'afficheur.

— Rechercher le repère R_1 et mettre le magnétophone en marche dans sa fonction lecture de cassette (play).

— Un peu avant le repère R_2, appuyer sur la touche P pour lancer le programme de lecture.

— Quand le transfert en RAM de la série des N octets est terminé, l'afficheur d'adresse indique PrEt. On peut alors arrêter le magnétophone.

Remarque très importante

La restitution correcte du programme à la RAM suppose que le réglage de volume du magnétophone — c'est-à-dire finalement la valeur de crête du signal v_s délivré par le magnétophone — soit convenable. Pour effectuer le réglage, le meilleur moyen consiste à observer à l'oscilloscope le signal v issu de l'interface de lecture et appliqué à PA_4. Le réglage de volume est bon si ce signal est bien constitué de niveaux 0 V et 5 V, sans altérations parasites.

A défaut d'oscilloscope, on peut effectuer le réglage en utilisant un contrôleur universel en fonction mesure d'une tension continue. Si on lui soumet v, le réglage du volume est correct quand l'aiguille oscille autour de la position moyenne 2,5 V. Si l'indication demeure longtemps 0 V ou 5 V, il convient de modifier le réglage.

La constitution d'une bibliothèque de programmes ou d'une banque de données nécessite beaucoup d'ordre. Pour chaque série d'octets enregistrée, il est nécessaire de noter :

— la nature de cette série d'octets ;

— les repères R_1, R_2, R_3 ;

— le nombre N des octets de cette série (en hexadécimal si possible).

Chapitre 11

COMPLÉMENTS AU PROGRAMME MONITEUR

Le chapitre 5 a donné la fonction générale et le mode d'utilisation du moniteur dont la description matérielle et logicielle a fait l'objet des chapitres 6, 7 et 8.

Afin de faciliter encore le travail du programmeur, nous avons conçu cinq programmes complémentaires d'aide à la programmation. Ils viennent s'ajouter aux programmes de transfert entre mémoire et cassette, de test de virginité et de lecture ou de programmation d'une EPROM type 2716. Comme ceux-ci, chacun de ces cinq nouveaux programmes peut être appelé — après avoir introduit en mémoire, grâce au moniteur, les données qui lui sont nécessaires — en frappant son adresse sur le clavier puis en appuyant sur la touche P. Quand l'exécution du programme sélectionné est terminée, on revient automatiquement à la réinitialisation du moniteur, à l'affichage et à la scrutation du clavier : l'afficheur d'adresse indique PrEt (à recevoir une nouvelle adresse en vue d'une autre mission).

Le tableau XI. 1 donne la liste et les adresses des neuf programmes enregistrés en EPROM. Le tableau XI. 2 fournit la liste et les adresses de tous les sous-programmes standard figurant dans l'EPROM moniteur. On n'oubliera pas que lors de l'établissement d'un programme, il est possible d'utiliser non seulement l'un de ces sous-programmes mais aussi les sous-programmes spécifiques d'appel et de retour écrits eux aussi en EPROM moniteur (chapitre 7).

Nous voulons expliquer ici quelle est la fonction exacte des cinq nouveaux programmes et donner des indications permettant de comprendre leur conception.

I. Fonction et utilisation des programmes complémentaires

I. 1. Transfert d'une série d'octets d'une zone mémoire dans une autre (voir illustration figure X. 1)

a) *Adresse du programme :* 0318

b) *Données utilisées par le programme* (à écrire préalablement en RAM)

— adresse du premier octet à transférer : lignes 0C00 et 0C01

Programmes	Adresses
Test de virginité et lecture d'une EPROM 2716 entière	0200
Test de virginité et lecture d'une zone d'EPROM 2716	020A
Programmation d'une zone d'EPROM 2716	024A
Transfert d'une série d'octets de la mémoire à une cassette	027B
Transfert d'une série d'octets d'une cassette à la mémoire	02B0
Transfert d'une série d'octets d'une zone mémoire dans une autre	0318
Formation du nombre d'octets à transférer	0331
Translation d'un programme d'une unité.	0349
Conversion BCD→Binaire d'un octet	03BD
Conversion Binaire→BCD d'un octet	03D9

Tableau XI. 1 Programmes disponibles sur l'EPROM moniteur

Sous-programmes standard	Adresses
Temporisation	0100
Affichage	010D
Scrutation du clavier	0125
Décalage $CA_1 \leftarrow CA_2 \leftarrow CA_3 \leftarrow CA_4$	013B
Décalage $CA_5 \leftarrow CA_6$	0149
Mise à jour du registre d'adresse R (A)	0150
Mise à jour du registre de donnée R (D). 0	0167
Lecture de la mémoire et stockage des codes d'affichage de la donnée	017B
Stockage des codes d'affichage de la donnée contenue dans R (D). 0	01CC
Stockage des codes d'affichage de l'adresse contenue dans R (A)	0193
Chargement des registre d'adresses R(A) et R (1) et du compteur d'octet R (B)	01CF
Formation de l'octet d'adressage de poids faible de l'EPROM	0303
Conversion BDC→Binaire	03C5
Convesion Binaire→BCD	03E1

Tableau XI. 2. Sous-programmes standards disponibles sur l'EPROM moniteur

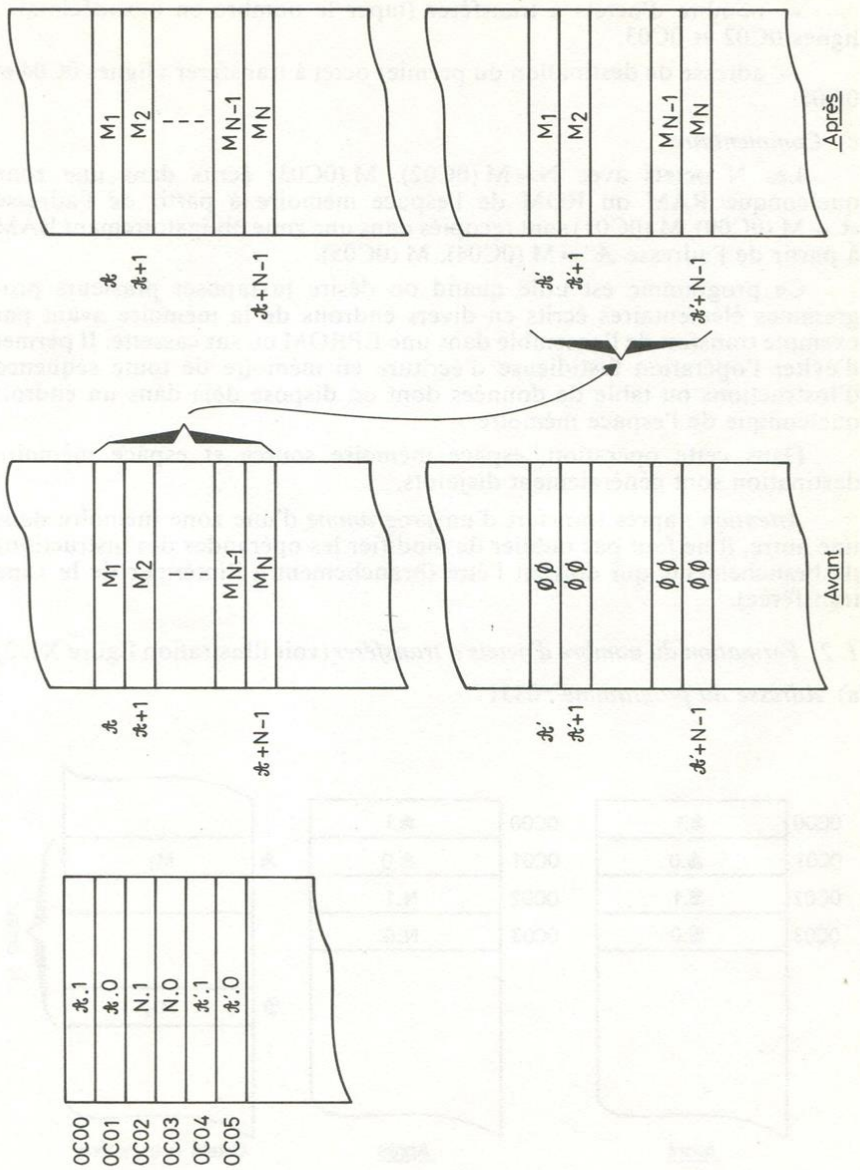


Fig. XI. 1. Fonction du programme de transfert d'une série d'octets d'une zone mémoire dans une autre

— nombre d'octets à transférer (taper le nombre en hexadécimal) : lignes 0C02 et 0C03.

— adresse de destination du premier octet à transférer : lignes 0C04 et 0C05

c) *Commentaire*

Les N octets avec N = M (0C02), M (0C03) écrits dans une zone quelconque RAM ou ROM de l'espace mémoire à partir de l'adresse $\mathcal{A}$ = M (0C00), M (0C01) sont recopiés dans une zone obligatoirement RAM à partir de l'adresse $\mathcal{A}'$ = M (0C04), M (0C05).

Ce programme est utile quand on désire juxtaposer plusieurs programmes élémentaires écrits en divers endroits de la mémoire avant par exemple transfert de l'ensemble dans une EPROM ou sur cassette. Il permet d'éviter l'opération fastidieuse d'écriture en mémoire de toute séquence d'instructions ou table de données dont on dispose déjà dans un endroit quelconque de l'espace mémoire.

Dans cette opération, espace mémoire source et espace mémoire destination sont généralement disjoints.

Attention : après transfert d'un *programme* d'une zone mémoire dans une autre, il ne faut pas oublier de modifier les opérandes des instructions de branchements qui doivent l'être (branchement à l'intérieur de la zone transférée).

I. 2. Formation du nombre d'octets à transférer (voir illustration figure XI. 2)

a) *Adresse du programme :* 0331

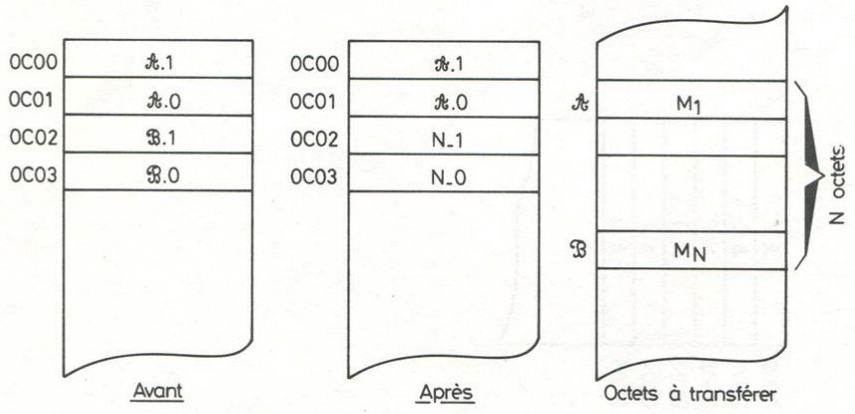


Fig. XI. 2. Fonction du programme de formation du nombre d'octets à transférer

b) *Données utilisées par le programme* (à écrire préalablement en RAM)

— adresse du premier octet à transférer : lignes 0C00 et 0C01
— adresse du dernier octet à transférer : lignes 0C02 et 0C03.

c) *Commentaire*

A partir de ces données, le programme permet le calcul automatique du nombre d'octets à transférer $N = \mathcal{B} - \mathcal{A} + 1$. Le nombre binaire N se charge dans la mémoire RAM aux adresses 0C02 et 0C03.

Le programme peut être utilisé lorsque la détermination en hexadécimal du nombre des octets à transférer n'est pas immédiate. Avant l'opération de transfert quelconque, les lignes 0C00 à 0C03 sont chargées grâce au moniteur par les adresses du premier et du dernier octet à transférer ; ce programme de calcul est exécuté. On peut alors lancer le programme de transfert.

De façon annexe, ce programme peut être utilisé pour trouver le résultat, sous forme hexadécimal, de la soustraction de deux nombres binaires de 16 bits : le premier opérande est placé à 0C02 à 0C03, le second à 0C00 et 0C01 ; le présent programme de calcul est lancé ; le résultat augmenté d'une unité peut être lu aux adresses 0C02 et 0C03, toujours grâce au moniteur (l'utilisateur n'ayant plus pour seul exercice mental qu'à ôter l'unité en excès).

I. 3. Translation d'un programme d'une unité (voir illustration figure XI. 3)

a) *Adresse du programme :* 0349.

b) *Données utilisées par le programme* (à écrire préalablement en RAM)

— adresse du code de la première instruction du programme à translater : lignes 0C00 et 0C01.

— nombre d'octets (en hexadécimal) du programme à translater : ligne 0C03

c) *Commentaire*

Quelle est l'utilité d'un tel programme de translation ? Lors de la mise au point d'un programme, on est souvent confronté au problème de la modification d'une séquence d'instructions (pour corriger une erreur ou réparer un oubli). Deux cas peuvent se présenter :

— la modification conduit à remplacer la séquence initiale par une séquence plus courte (c'est-à-dire utilisant moins d'espace mémoire). Il suffit alors de combler le « trou » laissé, par les codes d'instructions sans effets (NOP, REQ si Q est déjà à 0, SEX R (2) si X est déjà à 2...) ou bien de programmer un saut (SKP, LSKP) ou un branchement à la suite non modifiée du programme. Cette manipulation simple permet d'éviter la translation vers le haut de toute la suite du programme, opération longue et

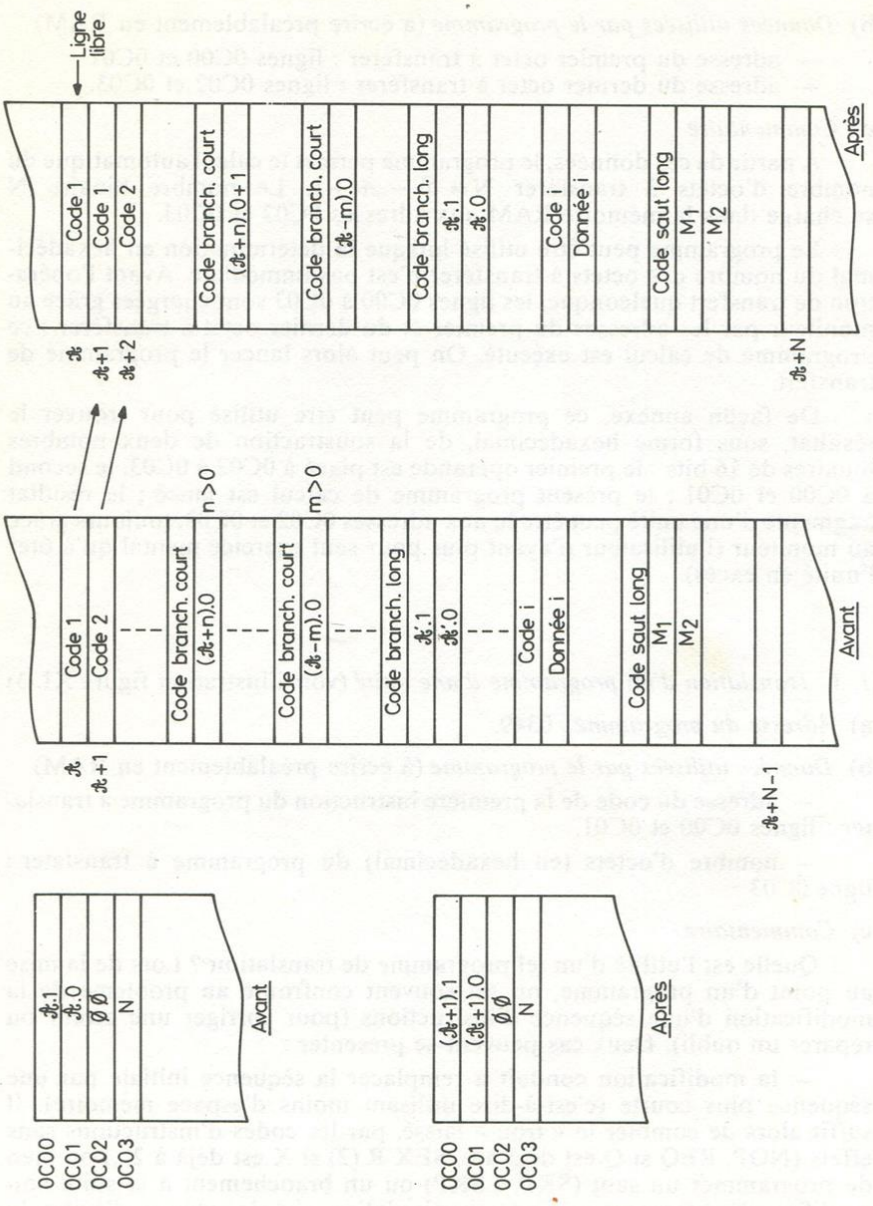


Fig. XI. 3. Fonction du programme de translation d'une unité

fastidieuse (même en s'aidant du moniteur) si cette suite est de taille importante ;

— la modification conduit à remplacer la séquence par une séquence plus longue. C'est malheureusement le cas le plus fréquent (à cause des oublis). Il oblige à translater la suite du programme vers le bas d'une ou plusieurs lignes. N'oublions pas que cette translation nécessite en plus la modification des opérandes des instructions de branchement à une adresse de la partie translatée.

Comme l'expérience de la programmation nous a souvent placé dans cette situation peu agréable, nous avons imaginé un programme qui réalise automatiquement, mais *seulement à l'intérieur d'une même page de mémoire,* la translation d'une unité d'adresse vers le bas de N = M (0C03) octets de *programme* (ce programme ne s'applique pas à la translation d'une table de données) enregistrés à partir de l'adresse $\mathcal{A}$ = M (0C00), M (0C01). Ce programme se charge de la modification des opérandes de branchements courts au-delà de l'adresse $\mathcal{A}$. L'exécution de ce programme incrémente automatiquement l'adresse $\mathcal{A}$ contenue aux lignes 0C00 et 0C01 de telle sorte qu'un nouveau lancement de ce programme translate une nouvelle fois d'une unité la même partie de programme en modifiant les opérandes de branchements courts.

Suggestion : si l'on désire translater de n lignes vers le bas toute une partie de programme, on peut avantageusement utiliser la touche U :

— charger en 0C74 et 0C75 l'adresse 0349 du programme de translation qui devient alors le programme spécifique de l'utilisateur ;

— charger en 0C00, 0C01 et 0C03 les données nécessaires au programme ;

— appuyer n fois sur la touche U pour obtenir la translation de n lignes et le calcul des adresses de branchement.

d) *Directives d'emploi du programme de translation*

— la translation ne se fait qu'à l'intérieur d'une même page de mémoire (l'octet de poids fort des adresses ne change pas) ;

— après utilisation de ce programme ne pas oublier de modifier dans le reste du programme non translaté les opérandes des branchements à des adresses de la partie de programme translatée ;

— le programme ne modifie jamais les opérandes des branchements longs, pas plus que les deux octets qui suivent les codes des sauts longs ;

— dans le cas peu fréquent où le code D4 de l'appel à un sous-programme est suivi, en plus des deux octets d'adresse du sous-programme, par des données, destinées à être utilisées par le sous-programme, il faut (sous peine de risque d'erreur) proscrire l'emploi du programme de translation car celui-ci peut interpréter une donnée comme étant un code. En étant vigilant, on peut malgré tout effectuer la translation en trois étapes : translation de la partie suivant les données liées à D4 ; translation de la

partie jusqu'à D4 et ses deux octets d'adresse du S.P. ; écriture des données destinées au S.P.

I. 4. Conversion BCD → Binaire

a) *Adresse du programme :* 03BD

b) *Donnée utilisée par le programme :* octet à convertir écrit préalablement à l'adresse 0C06.

c) *Commentaire*

Le programme effectue la conversion BCD → Binaire de cet octet et écrit le résultat à l'adresse initiale 0C06.

Au niveau de l'utilisateur, c'est en fait la conversion décimal → hexadécimal qui est effectuée. On sait que la machine microprocesseur ne connaît que le langage binaire. L'utilisateur communique normalement avec elle en hexadécimal par l'intermédiaire du clavier et de l'afficheur : c'est le système VILÉMIO qui est chargé de la traduction hexadécimal ↔ binaire. Si l'usage de l'hexadécimal ne présente pas de difficultés au niveau des adresses ou des codes d'instructions, il n'en est pas toujours de même en ce qui concerne les données. Le système décimal s'impose : les données parviennent ou sont connues souvent sous forme décimale. Il est possible d'effectuer une conversion décimal-hexadécimal avant de les fournir au système ou bien on peut confier au système lui-même la mission d'effectuer la conversion (c'est précisément le but de ce programme).

Introduisons grâce au clavier la donnée $\alpha\,\beta$ décimale à deux chiffres (0 à 9) à l'adresse 0C06 de la mémoire, comme nous le ferions si $\alpha\,\beta$ était un nombre hexadécimal. Ce qui s'incrit dans la ligne 0C06, c'est la suite des deux quartets associés aux chiffres hexadécimaux α et β car le système a opéré la conversion en binaire du nombre introduit de la même manière que s'il s'agissait d'un nombre hexadécimal. Le résultat M (0C06) obtenu est le nombre Décimal Codé en Binaire (BCD). Le programme proposé rétablit la forme binaire naturelle de ce nombre, seule exploitable par le microprocesseur. Si antérieurement à la conversion on procède à la lecture du nombre rangé à 0C06, l'afficheur indique sa traduction sous forme hexadécimale. On a frappé initialement le nombre décimal, on lit finalement le même nombre dans le système hexadécimal. De l'extérieur, on a effectué la conversion décimal → hexadécimal.

Le programme de conversion fait appel lui-même à un sous-programme écrit à partir de l'adresse 03C5 et qui effectue la conversion BCD → Binaire du contenu d'une ligne mémoire quelconque d'adresse $\mathcal{A}$. L'appel de ce sous-programme s'effectue bien entendu grâce à l'instruction SEP R (4) dont le code D4 est suivi des deux octets 03 et C5 de l'adresse du sous-programme puis des deux octets de l'adresse $\mathcal{A}$ où doit s'effectuer la conversion. On l'utilisera particulièrement lorsqu'une donnée parvient d'un périphérique sous forme BCD et est introduite dans la mémoire grâce à une

instruction INPUT ou un DMA-IN (il faut la convertir en binaire avant de l'utiliser).

I. 5. Conversion Binaire→ BCD

a) *Adresse du programme :* 03D9

b) *Donnée utilisée par le programme :* octet à convertir écrit préalablement à l'adresse 0C06

c) *Commentaire*

Le programme effectue la conversion Binaire → BCD de cet octet et écrit le résultat à l'adresse 0C06.

C'est la conversion inverse de la précédente. Au niveau de l'utilisateur, c'est la conversion hexadécimal → décimal qui est effectuée.

Attention : pour $N_{Hexa} \leq 63$ $N_{Dec} \leq 99$
pour $64 \leq N_{Hexa} \leq C7$ $100 \leq N_{Dec} \leq 199$
pour $C8 \leq N_{Hexa} \leq FF$ $200 \leq N_{Dec} \leq 255$

Si $N_{Hexa} \geq 64$, le nombre décimal correspondant se traduit en BCD sur plus d'un octet. A l'issue du programme de conversion, la retenue éventuelle 1 ou 2 est disponible sur le registre R (9).1.

Ce programme de conversion fait appel à un sous-programme écrit à partir de l'adresse 03E1 qui effectue la conversion Binaire → BCD du contenu de la ligne mémoire d'adresse $\mathcal{A}$ précisée après le code D4 d'appel et l'adresse 03E1.

II. Conception des programmes complémentaires

II. 1. Transfert d'une série d'octets d'une zone mémoire dans une autre

(se référer conjointement à l'organigramme de la figure XI. 4 et au tableau XI. 3)

Il est d'abord fait appel au sous-programme de chargement des registres d'adresse et du compteur d'octets : R (A) reçoit l'adresse $\mathcal{A}$ = M (0C00), M (0C01) du premier octet à transférer, R (B) reçoit le nombre N = M (0C02), M (0C03) des octets à transférer et R (1) est chargé par l'adresse $\mathcal{A}'$ = M (0C04), M (0C05) de destination du premier octet.

Chaque octet est transféré de la zone source vers la zone destination en passant bien entendu par l'accumulateur, et les pointeurs R (A) et R (1) sont incrémentés. Grâce au bouclage et à l'utilisation du compteur R (B), cette opération est effectuée N fois.

II. 2. Formation du nombre d'octets à transférer (voir organigramme de la figure XI. 5 et tableau XI. 4)

Il s'agit d'effectuer des opérations sur des octets situés dans la RAM

Adresses	Codes et opérandes	Mnémoniques	Commentaires
0318	D4 01CF	SEP R (4)	Appel du S.P. de chargement des registres d'adresse et du compteur d'octets
1B	4A	LDA R (A)	Chargement de l'octet et incrémentation du pointeur
1C	51	STR R (1)	Stockage de l'octet à sa destination...
1D	11	INC R (1)	... et incrémentation du pointeur
1E	2B	DEC R (B)	Décrémentation du compteur
1F	9B	GHI R (B)	Branchement pour reprendre le transfert d'un octet si la liste n'est pas épuisée
20	3A 1B	BNZ 1B	
22	8B	GLO R (B)	
23	3A 1B	BNZ 1B	
25	C0 0037	LBR	Branchement à l'initialisation du moniteur.

Tableau XI. 3. Programme de transfert d'une zone mémoire dans une autre

Adresses	Codes et opérandes	Mnémoniques	Commentaires
0331	F8 01	LDI 01	Initialisation de R (F) à 0C01 et désignation comme pointeur
33	AF	PLO R (F)	
35	EF	SEX R (F)	
36	72	LDXA	Chargement dans D de $\mathcal{A}$.0 = M (0C01) et mise de R (F) à 0C03
37	60	IRX	
38	F5	SD	Formation de $\mathcal{B}$.0– $\mathcal{A}$.0 dans D. retenue dans DF
39	AB	PLO R (B)	Sauvegarde dans R (B). 0
3A	2F	DEC R (F)	Mise de R (F) à 0C00
3B	2F	DEC R (F)	
3C	2F	DEC R (F)	
3D	72	LDXA	Chargement dans D et $\mathcal{A}$ .1 = M (0C00) et mise de R (F) à 0C02
3E	60	IRX	
3F	75	SDB	Formation de $\mathcal{B}$ .1– $\mathcal{A}$.1 dans D en tenant compte de la retenue et sauvegarde dans R (B).1
40	BB	PHI R (B)	
41	1B	INC R (B)	Le contenu de R (B) passe à N
42	9B	GHI R (B)	Stockage du résultat N en M (0C02) en M (0C03)
43	5F	STR R (F)	
44	1F	INC R (F)	
45	8B	GLO R (B)	
46	5F	STR R (F)	
47	C0 0037	LBR	Branchement à l'initialisation du moniteur

Tableau XI. 4. Programme de formation du nombre d'octets à transférer

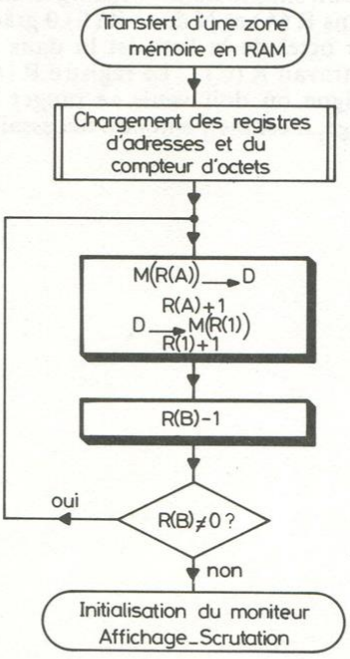


Fig. XI. 4. Organigramme du programme de transfert d'une zone mémoire en RAM

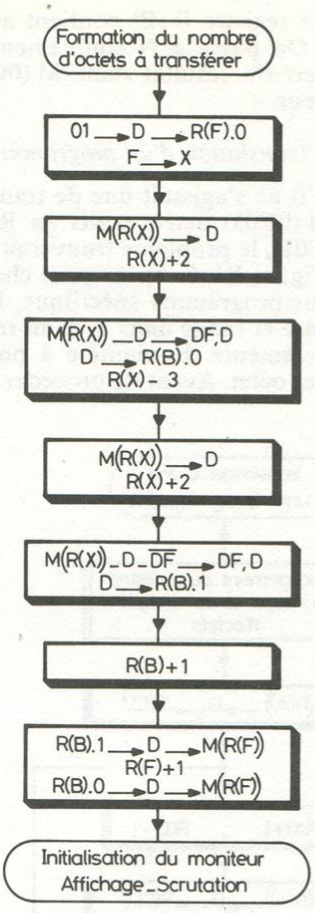


Fig. XI. 5. Organigramme du programme de formation du nombre d'octets à transférer

aux adresses 0C00 à 0C04 : il est nécessaire de désigner un pointeur par l'intermédiaire du registre X. R (F) est naturellement choisi comme pointeur de cette zone et il est initialisé à 0C01 (l'initialisation de R (F).1 a été effectuée par le moniteur). M (0C01) = $\mathcal{A}$ .0 est chargé dans D pour être soustrait de M (0C03) = $\mathcal{B}$ .0. Le résultat formé dans D est sauvegardé dans R (B).0 alors que DF conserve la retenue éventuelle de l'opération. M (0C00) = $\mathcal{A}$ .1 est à son tour chargé dans D pour être soustrait de M (0C02) = $\mathcal{B}$ .1 en tenant compte de la retenue. Le résultat formé dans D est sauvegardé dans R (B).1.

Le registre R (B) contient alors le nombre N-1 (problème d'intervalles !). On passe à N simplement par incrémentation puis on effectue le transfert du résultat dans M (0C02) et M (0C03) avant branchement au moniteur.

II. 3. Translation d'un programme d'une unité

S'il ne s'agissait que de translater d'une ligne vers le bas une série de N = M (0C03) octets écrits en RAM à partir de l'adresse $\mathcal{A}$ = M (0C00), M (0C01), le problème trouverait une solution simple dans l'organigramme de la figure XI. 6. Après avoir chargé $\mathcal{A}$ dans R (A) et N dans R (B).0 grâce au sous-programme spécifique, le premier octet de la liste est lu dans la mémoire et rangé dans le demi-registre de travail R (C).1. Le registre R (A) est incrémenté de manière à pointer la ligne où doit venir se ranger le premier octet. Avant de procéder au stockage, il est bien entendu nécessaire

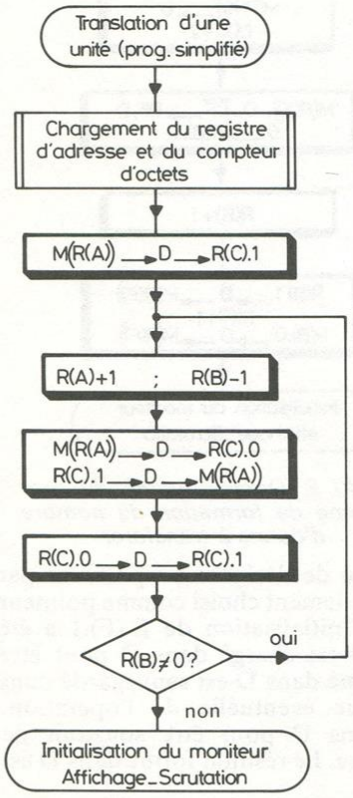


Fig. XI. 6. Organigramme d'un programme simple de translation d'une unité

d'effectuer la lecture du deuxième octet qui est sauvegardé provisoirement dans R (C).0. Le stockage du premier octet est alors effectué à la ligne suivante puis le deuxième octet est transféré dans R (C).1. Ainsi R (C).0 est prêt à recevoir, pour sa sauvegarde, le troisième octet. Ce processus se reproduit autant de fois qu'il y a d'octets à translater ; le registre R (B).0 utilisé en compteur de boucles permet de savoir quand la liste est épuisée.

Le problème que nous envisageons de résoudre est plus complexe : on désire que le programme sache reconnaître parmi les octets de la séquence de *programme* à translater ceux qui sont les *codes* de branchements courts (c'est-à-dire dans la même page) et qu'il agisse de la manière suivante :

— si l'opérande (faible poids de l'adresse de branchement) est inférieur à $\mathcal{A}$ .0 il doit être simplement translaté d'une unité, sans lui apporter de modification puisque à l'issue de la translation la partie de programme à laquelle ce branchement renvoie n'aura pas bougé ;

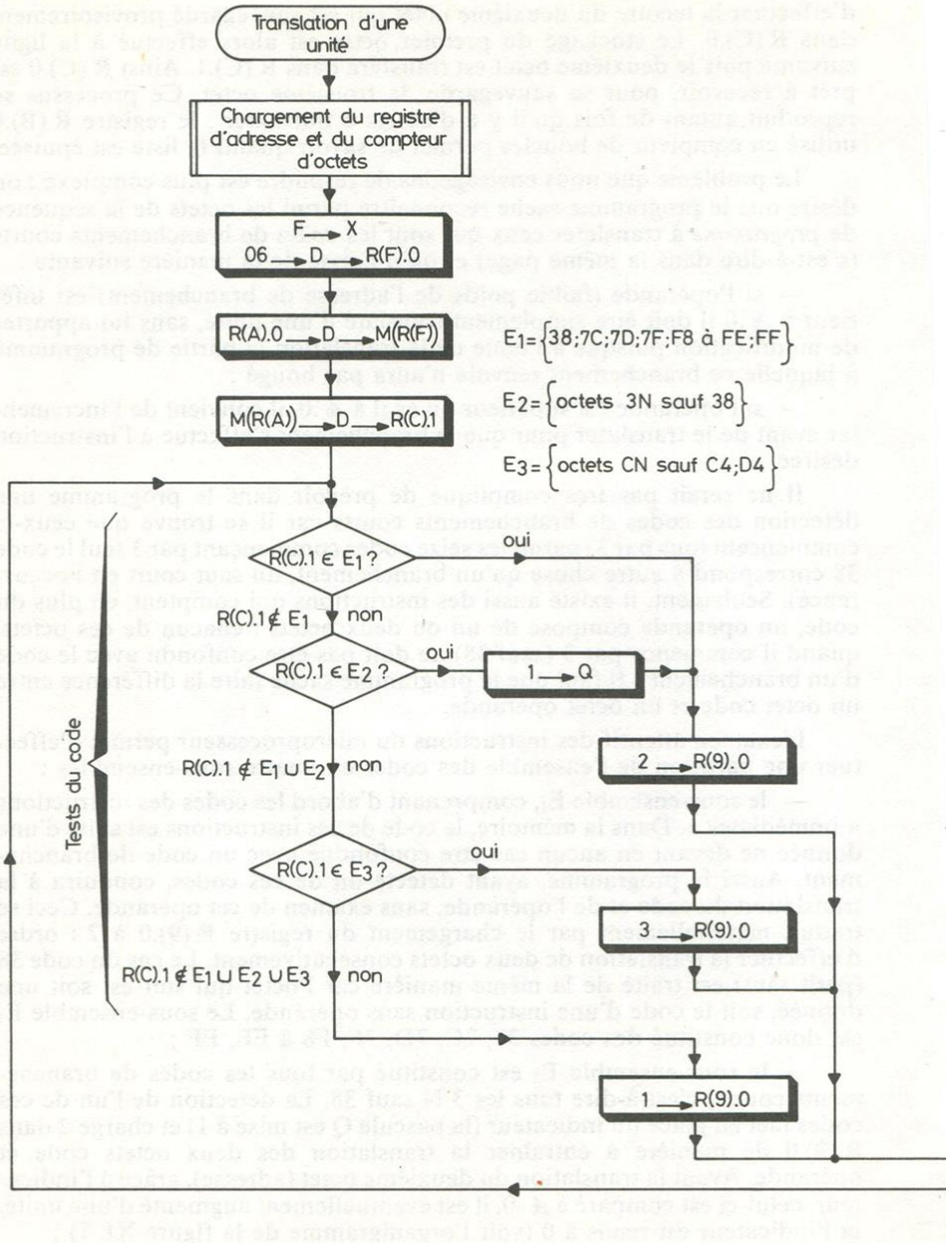
— si l'opérande est supérieur ou égal à $\mathcal{A}$ .0, il convient de l'incrémenter avant de le translater pour que le branchement s'effectue à l'instruction désirée.

Il ne serait pas très compliqué de prévoir dans le programme une détection des codes de branchements courts car il se trouve que ceux-ci commencent tous par 3 (parmi les seize codes commençant par 3 seul le code 38 correspond à autre chose qu'un branchement, un saut court en l'occurrence). Seulement, il existe aussi des instructions qui comptent, en plus du code, un opérande composé de un ou deux octets : chacun de ces octets, quand il commence par 3 (sauf 38) ne doit pas être confondu avec le code d'un branchement ! Il faut que le programme sache faire la différence entre un octet code et un octet opérande.

L'examen attentif des instructions du microprocesseur permet d'effectuer une partition de l'ensemble des codes en quatre sous-ensembles :

— le sous-ensemble E_1, comprenant d'abord les codes des instructions « immédiates ». Dans la mémoire, le code de ces instructions est suivi d'une donnée ne devant en aucun cas être confondue avec un code de branchement. Aussi le programme, ayant détecté un de ces codes, conduira à la translation du code et de l'opérande, sans examen de cet opérande. Ceci se traduit matériellement par le chargement du registre R (9).0 à 2 : ordre d'effectuer la translation de deux octets consécutivement. Le cas du code 38 (petit saut) est traité de la même manière car l'octet qui suit est soit une donnée, soit le code d'une instruction sans opérande. Le sous-ensemble E_1 est donc constitué des codes 38, 7C, 7D, 7F, F8 à FE, FF ;

— le sous-ensemble E_2 est constitué par tous les codes de branchements courts, c'est-à-dire tous les 3 N sauf 38. La détection de l'un de ces codes met en place un indicateur (la bascule Q est mise à 1) et charge 2 dans R (9).0 de manière à entraîner la translation des deux octets code et opérande. Avant la translation du deuxième octet (adresse), grâce à l'indicateur, celui-ci est comparé à $\mathcal{A}$ .0, il est éventuellement augmenté d'une unité, et l'indicateur est remis à 0 (voir l'organigramme de la figure XI. 7) ;


Translation d'une unité
Chargement du registre d'adresse et du compteur d'octets
F → X
06 → D → R(F).0
R(A).0 → D → M(R(F))
M(R(A)) → D → R(C).1
E1 = {38;7C;7D;7F;F8 à FE;FF}
E2 = {octets 3N sauf 38}
E3 = {octets CN sauf C4;D4}
R(C).1 ∈ E1 ?
oui
R(C).1 ∉ E1
non
R(C).1 ∈ E2 ?
oui
1 → Q
2 → R(9).0
R(C).1 ∉ E1 ∪ E2
non
R(C).1 ∈ E3 ?
oui
3 → R(9).0
R(C).1 ∉ E1 ∪ E2 ∪ E3
non
1 → R(9).0
Tests du code

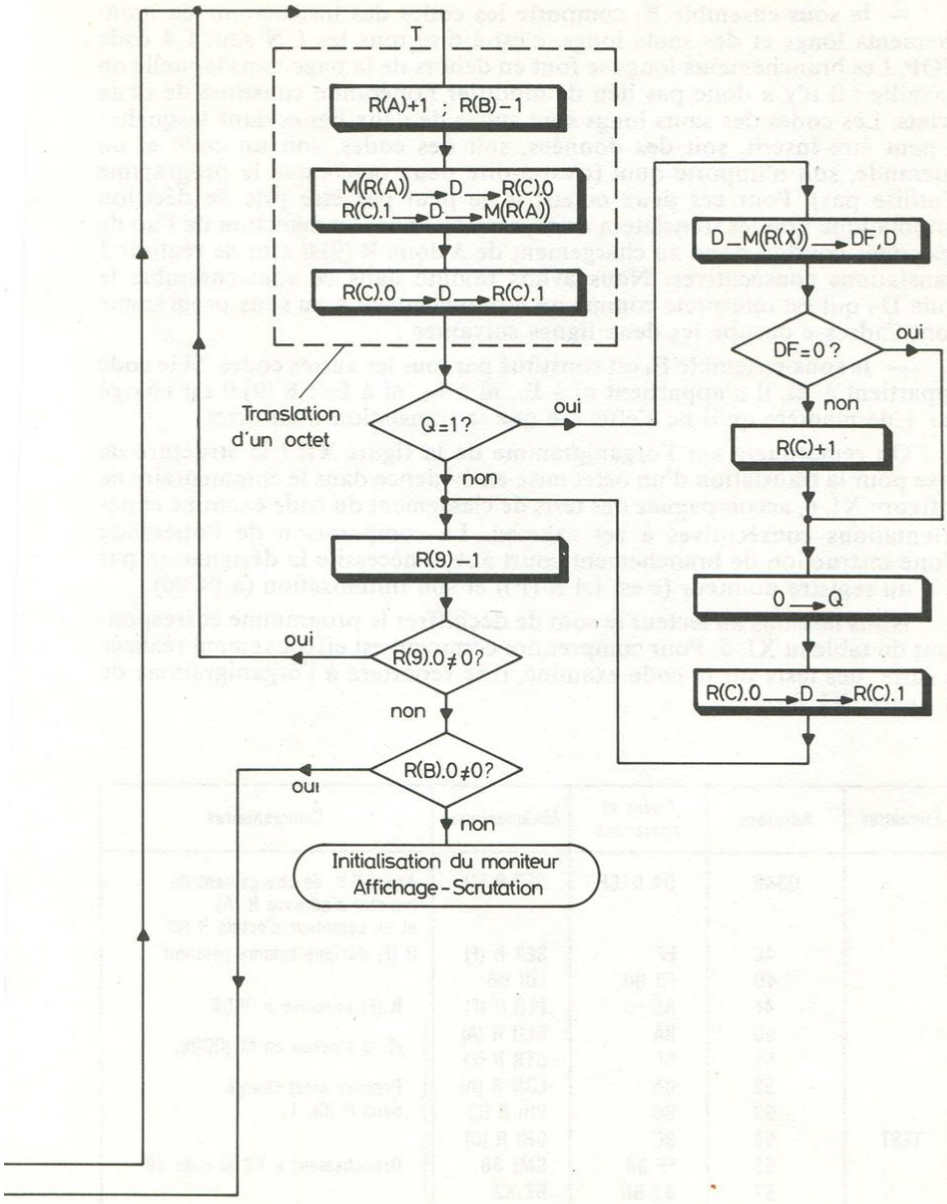


Fig. XI. 7. Organigramme du programme de translation d'une unité

— le sous-ensemble E_3 comporte les codes des instructions de branchements longs et des sauts longs, c'est-à-dire tous les CN sauf C4 code NOP. Les branchements longs se font en dehors de la page dans laquelle on travaille : il n'y a donc pas lieu de modifier l'opérande constitué de deux octets. Les codes des sauts longs sont suivis de deux lignes dans lesquelles il peut être inscrit, soit des données, soit des codes, soit un code et un opérande, soit n'importe quoi (c'est-à-dire deux octets que le programme n'utilise pas). Pour ces deux octets, il ne peut pas être pris de décision automatique ; on les translatera donc, sans examen. La détection de l'un de ces codes conduit donc au chargement de 3 donc R (9).0 afin de réaliser 3 translations consécutives. Nous avons rajouté dans ce sous-ensemble le code D4 qui est interprété comme un « branchement » au sous-programme dont l'adresse occupe les deux lignes suivantes ;

— le sous-ensemble E_4 est constitué par tous les autres codes. Si le code appartient à E_4, il n'appartient ni à E_1, ni à E_2, ni à E_3 ; R (9).0 est chargé par 1 de manière qu'il ne s'effectue que la translation d'un octet.

On remarquera sur l'organigramme de la figure XI. 7 la structure de base pour la translation d'un octet mise en évidence dans le commentaire de la figure XI. 6, accompagnée des tests de classement du code examiné et des orientations consécutives à cet examen. La comparaison de l'opérande d'une instruction de branchement court à $\mathcal{A}$ .0 nécessite la désignation par X d'un registre pointeur (c'est ici R (F)) et son initialisation (à 0C06).

Nous laissons au lecteur le soin de déchiffrer le programme correspondant du tableau XI. 5. Pour comprendre comment est effectivement réalisée la suite des tests sur le code examiné, il se reportera à l'organigramme de la figure XI. 8.

Étiquettes	Adresses	Codes et opérandes	Mnémoniques	Commentaires
	0349	D4 01CF	SEP R (4)	Appel S.P. de chargement du registre d'adresse R (A) et du compteur d'octets R (B)
	4C	EF	SEX R (F)	R (F) désigné comme pointeur
	4D	F8 06	LDI 06	
	4F	AF	PLO R (F)	R (F) initialisé à 0C06
	50	8A	GLO R (A)	$\mathcal{A}$ .0 stockée en M (0C06)
	51	5F	STR R (F)	
	52	0A	LDN R (A)	Premier octet chargé dans R (C). 1
	53	BC	PHI R (C)	
TEST	54	9C	GHI R (C)	Branchement à X2 si code 38
	55	FF 38	SMI 38	
	57	32 B0	BZ X2	

Étiquettes	Adresses	Codes et opérandes	Mnémoniques	Commentaires
	59	9C	GHI R (C)	
	5A	FF 7C	SMI 7C	Branchement à X2 si code 7C
	5C	32 B0	BZ X2	
	5E	9C	GHI R (C)	
	5F	FF 7D	SMI 7D	Branchement à X2 si code 7D
	61	32 B0	BZ X2	
	63	9C	GHI R (C)	
	64	FF 7F	SMI 7F	Branchement à X2 si code 7F
	66	32 B0	BZ X2	
	68	9C	GHI R (C)	
	69	FF FE	SMI FE	Branchement à X1 si code FE
	6B	32 8A	BZ X1	
	6D	9C	GHI R (C)	
	6E	FF F8	SMI F8	Branchement à X2 si code $\geqslant$F8 sauf FE
	70	33 B0	BDF X2	
	72	9C	GHI R (C)	
	73	FA F0	ANI F0	Branchement à SEQ si code 3N sauf 38
	75	FF 30	SMI 30	
	77	32 BA	BZ SEQ	
	78	9C	GHI R (C)	
	7A	FF D4	SMI D4	Branchement à X3 si code D4
	7C	32 B5	BZ X3	
	7E	9C	GHI R (C)	
	7F	FF C4	SMI C4	Branchement à X1 si code C4
	81	32 8A	BZ X1	
	83	9C	GHI R (C)	
	84	FA F0	ANI F0	Branchement à X3 si code CN sauf C4
	86	FF C0	SMI C0	
	88	32 B5	BZ X3	
	8A	F8 01	LDI 01	Mise à 1 de R (9). 0
X1	8C	A9	PLO R (9)	
TRANS	8D	1A	INC R (A)	
	8E	2B	DEC R (B)	
	8F	0A	LDN R (A)	
	90	AC	PLO R (C)	Translation d'un octet
	91	9C	GHI R (C)	
	92	5A	STR R (A)	
	93	8C	GLO R (C)	
	94	BC	PHI R (C)	
	95	31 A6	BQ EXA	Branch. à examen de l'adresse si indicateur mis
	97	29	DEC R (9)	Gestion du nombre de translations à effectuer
	98	89	GLO R (9)	
	99	3A 8D	BNZ TRANS	

Étiquettes	Adresses	Codes et opérandes	Mnémoniques	Commentaires
	9B	8B	GLO R (B)	Retour à tests si liste non épuisée
	9C	3A 54	BNZ TEST	
	9E	F8 01	LDI 01	Incrémentation de $\mathcal{A}$.0 dans M (0C01)
	A0	AF	PLO R (F)	
	A1	F4	ADD	
	A2	5F	STR R (F)	
	A3	C0 0037	LBR	Retour au moniteur
EXA	A6	9C	GHI R (C)	Examen de l'adresse de branchement, incrémentaton s'il y a lieu, remise de Q à 0 transfert de l'adresse dans R (C). 1
	A7	F7	SM	
	A8	3B AB	BNF	
	AA	1C	INC R (C)	
	AB	7A	REQ	
	AC	8C	GLO R (C)	
	AD	BC	PHI R (C)	
	AE	30 97	BR 97	
X2	B0	F8 02	LDI 02	Mise à 2 de R (9). 0 et branch. pour translation
	B2	A9	PLO R (9)	
	B3	30 8D	BR TRANS	
	B5	F8 03	LDI 03	Mise à 3 de R (9) et branch. pour translation
	B7	A9	PLO R (9)	
	B8	30 8D	BR TRANS	
SEQ	BA	7B	SEQ	Mise de Q à 1 et branch à X2
	BB	30 B0	BR X2	

Tableau XI. 5. Programme de translation d'une unité

Remarque

Là encore, nous savons que ce programme peut être amélioré. Nous nous sommes interdit l'emploi « d'astuces » de programmation, qui demandent trop d'explications, pour n'exposer que l'essentiel.

II. 4. Conversion BCD → Binaire

Le programme ne fait qu'appeler le sous-programme de conversion BCD → Binaire et réaliser ensuite le branchement au moniteur (voir tableau XI.6.). Nous trouvons ici une deuxième illustration intéressante de la possibilité qu'offre la méthode de gestion des sous-programmes standard de faire « passer » une donnée du programme appelant au sous-programme. C'est l'adresse de la ligne mémoire dans laquelle doit se faire la conversion BCD → Binaire qui est placée, dans le programme appelant, juste après l'adresse 03E1 du sous-programme. Le sous-programme (organigramme de

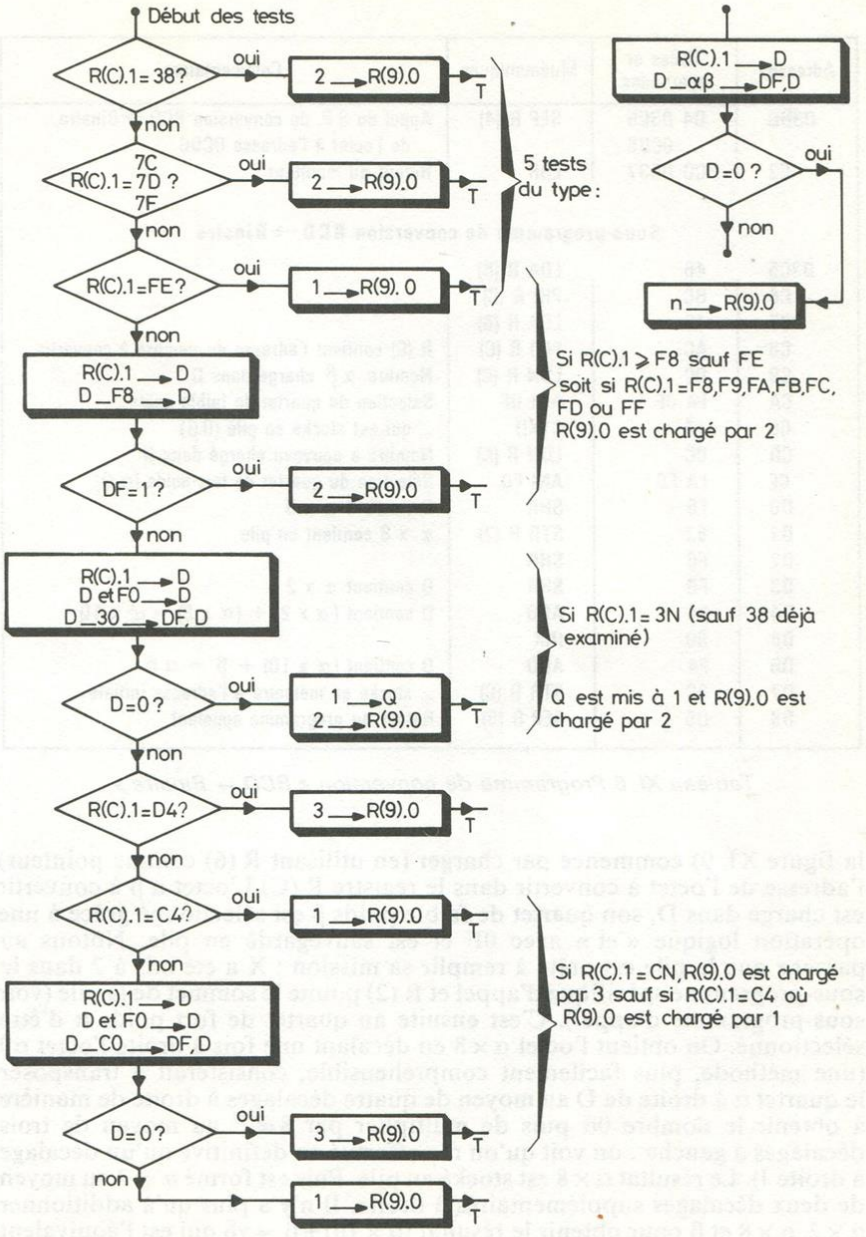


Fig. XI. 8. Organigramme de réalisation des tests

Adresses	Codes et opérandes	Mnémoniques	Commentaires
03BD	D4 03C5	SEP R (4)	Appel du S.P. de conversion BCD→Binaire...
	0C06		... de l'octet à l'adresse 0C06
C2	C0 0037	LBR	Retour au moniteur
		Sous-programme de conversion BCD→Binaire	
03C5	46	LDA R (6)	
C6	BC	PHI R (C)	
C7	46	LDA R (6)	
C8	AC	PLO R (C)	R (C) contient l'adresse du nombre à convertir
C9	0C	LDN R (C)	Nombre $\alpha\beta$ chargé dans D
CA	FA 0F	ANI 0F	Sélection du quartet de faible poids...
CC	73	STXD	... qui est stocké en pile (0β)
CD	0C	LDN R (C)	Nombre à nouveau chargé dans D
CE	FA F0	ANI F0	Sélection du quartet de fort poids (α 0)
D0	F6	SHR	D contient α x 8
D1	52	STR R (2)	α x 8 contient en pile
D2	F6	SHR	
D3	F6	SHR	D contient α x 2
D4	F4	ADD	D contient (α x 2) + (α x 8) = α x 10
D5	60	IRX	
D6	F4	ADD	D contient (α x 10) + β = $\alpha\delta$
D7	5C	STR R (C)	... stocké en mémoire à l'adresse initiale
D8	D5	SEP R (5)	Retour au programme appelant

Tableau XI. 6 Programme de conversion « BCD → Binaire »

la figure XI. 9) commence par charger (en utilisant R (6) comme pointeur) l'adresse de l'octet à convertir dans le registre R (C).L'octet $\alpha\beta$ à convertir est chargé dans D, son quartet de faible poids β est sélectionné grâce à une opération logique « et » avec 0F et est sauvegardé en pile. Notons au passage que la pile est prête à remplir sa mission : X a été mis à 2 dans le sous-programme spécifique d'appel et R (2) pointe le sommet de la pile (voir sous-programme d'appel). C'est ensuite au quartet de fort poids α d'être sélectionné. On obtient l'octet $\alpha \times 8$ en décalant une fois à droite l'octet $\alpha 0$ (une méthode, plus facilement compréhensible, consisterait à transposer le quartet α à droite de D au moyen de quatre décalages à droite de manière à obtenir le nombre 0α puis de multiplier par $8 = 2^3$ au moyen de trois décalages à gauche : on voit qu'on n'a effectué en définitive qu'un décalage à droite !). Le résultat $\alpha \times 8$ est stocké en pile. Puis est formé $\alpha \times 2$ au moyen de deux décalages supplémentaires à droite. Il n'y a plus qu'à additionner $\alpha \times 2$, $\alpha \times 8$ et β pour obtenir le résultat $(\alpha \times 10) + \beta = \gamma\delta$ qui est l'équivalent naturel du nombre BCD $\alpha\beta$.

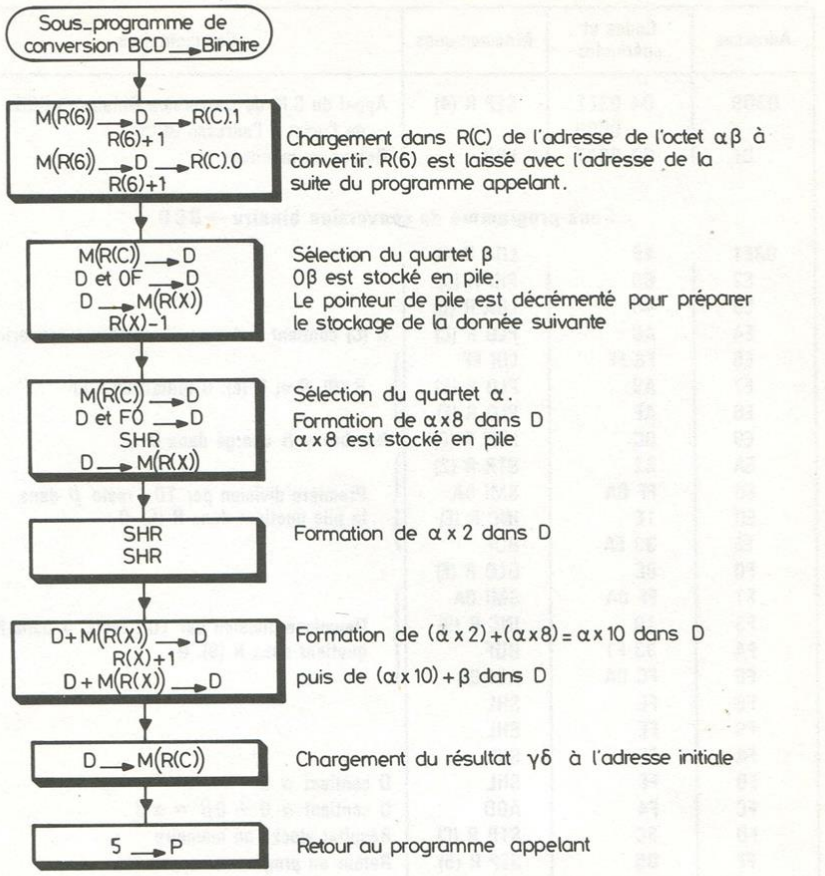


Fig. XI. 9. Organigramme du sous-programme de conversion BCD → Binaire

II. 5. Conversion Binaire → BCD

Là encore, le programme ne fait qu'appeler un sous-programme de conversion Binaire → BCD puis réaliser le branchement au moniteur (voir tableau XI. 7).

Le sous-programme commence par charger dans R (C) l'adresse de la mémoire où doit se faire la conversion. Le principe de cette conversion consiste à diviser le nombre à convertir par 10 (soit 0A en hexadécimal). Le reste de l'opération (c'est-à-dire le quartet β) est formé dans la pile. Le quotient formé dans R (E).0 est à nouveau divisé par 10. Le reste apparaît

Adresses	Codes et opérandes	Mnémoniques	Commentaires
03D9	D4 03E1	SEP R (4)	Appel du S.P. de conversion Binaire→BCD...
	0C06		... de l'octet à l'adresse 0C06
DE	C0 0037	LBR	Retour au moniteur
		Sous-programme de conversion binaire→BCD	
03E1	46	LDA R (6)	
E2	BC	PHI R (C)	
E3	46	LDA R (6)	
E4	AC	PLO R (C)	R (C) contient l'adresse du nombre à convertir
E5	F8 FF	LDI FF	
E7	A9	PLO R (9)	R (9). 0 et R (E). 0 initialisés à FF
E8	AE	PLO R (E)	
E9	0C	LDN R (C)	Nombre $\gamma\ \delta$ chargé dans D
EA	52	STR R (2)	
EB	FF 0A	SMI 0A	Première division par 10 : reste β dans
ED	1E	INC R (E)	la pile quotient dans R (E). 0
EE	33 EA	BDF	
F0	8E	GLO R (E)	
F1	FF 0A	SMI 0A	
F3	19	INC R (9)	Deuxième division par 10 : reste α dans D
F4	33 F1	BDF	quotient dans R (9). 0
F6	FC 0A	ADI 0A	
F8	FE	SHL	
F9	FE	SHL	
FA	FE	SHL	
FB	FE	SHL	D contient α 0
FC	F4	ADD	D contient $\alpha\ 0 + 0\beta = \alpha\beta$
FD	5C	STR R (C)	Résultat stocké en mémoire
FE	D5	SEP R (5)	Retour au programme appelant

Tableau XI. 7. Programme de conversion « Binaire → BCD »

dans D alors que le quotient (0, 1 ou 2) se forme dans R (9).0. Il ne reste plus qu'à regrouper les résultats c'est-à-dire construire $\alpha\beta$. Si une retenue existe (1 ou 2), elle est disponible sur R (9).0. Le détail des opérations figure sur l'organigramme de la figure XI. 10.

Note au lecteur :

Nous avons bien conscience de ce que la lecture et la compréhension de ces programmes est une opération qui peut rebuter. Il faut comprendre

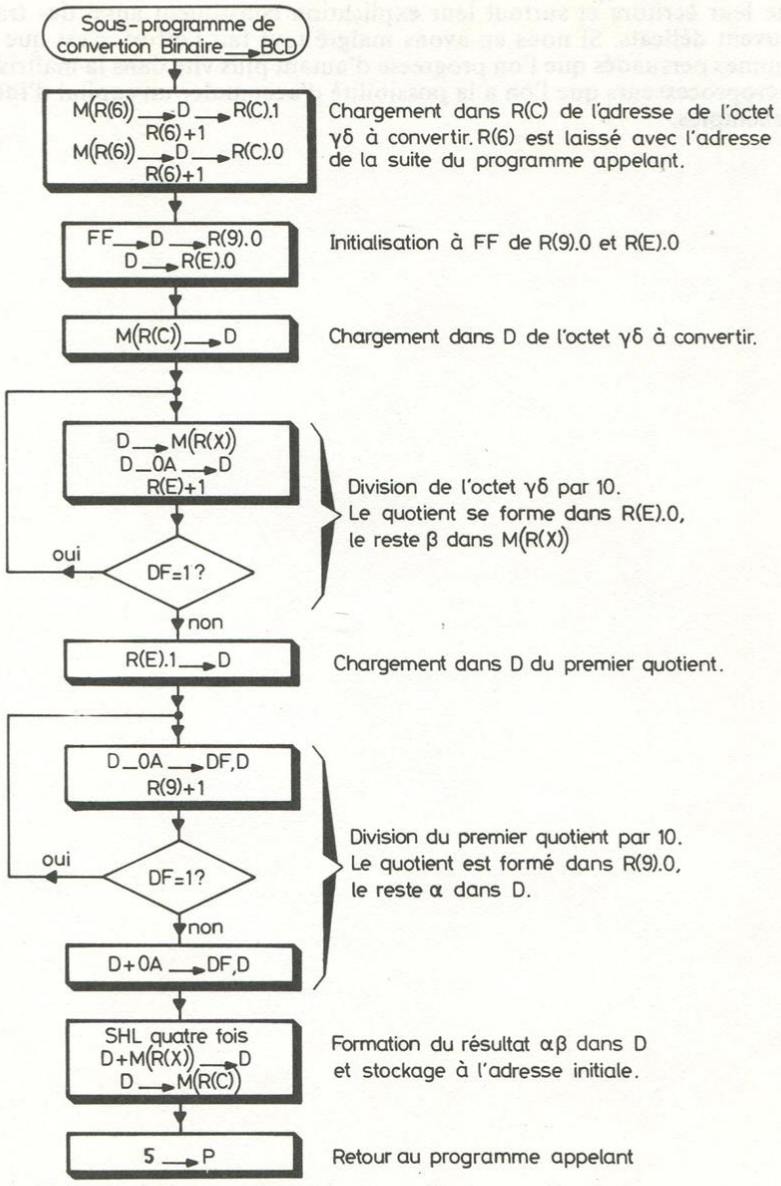
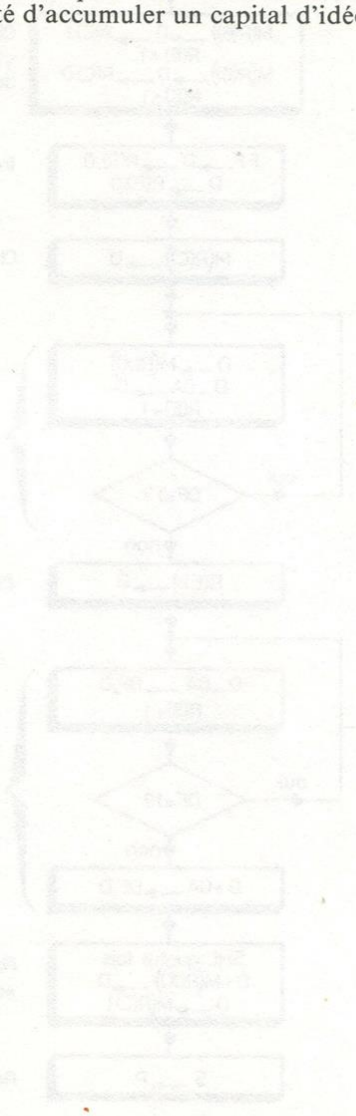


Fig. XI. 10. Organigramme du sous-programme de conversion Binaire → BCD

que leur écriture et surtout leur explication constituent aussi des travaux souvent délicats. Si nous en avons malgré tout fait l'effort, c'est que nous sommes persuadés que l'on progresse d'autant plus vite dans la maîtrise des microprocesseurs que l'on a la possibilité d'accumuler un capital d'idées et d'exemples.



INTRODUCTION
A LA TROISIÈME PARTIE

I. Utilisation du système VILÉMIO

Le lecteur qui a réalisé les maquettes VILÉMIO 1, 2 et 3 (maquettes « processeur », « mise au point », et « clavier-afficheur ») dispose d'un système qui possède une vie propre, et qu'il peut utiliser même s'il ne désire pas mettre au point une application. Cependant, les possibilités de ce système sont nettement augmentées par l'adjonction de la maquette VILÉMIO 4 (maquette « entrée/sortie ») : le système devient alors effectivement « micro-ordinateur ».

L'ensemble que nous proposons peut être *utilisé* selon essentiellement deux modes différents.

I. 1. Le mode « micro-ordinateur » MODE I

Dans ce mode, l'utilisateur considère son micro-ordinateur comme tel. C'est-à-dire que son but est de commander et contrôler des périphériques divers (de son choix) placés sur les entrées et sorties de VILÉMIO 4. Il faut noter que dans ce mode, la maquette VILÉMIO 3 constitue également un périphérique. La tâche de l'utilisateur est donc de concevoir des programmes de « gestion » de ces périphériques, qu'il essaiera en les écrivant dans la RAM 2 K située aux adresses de 0400 à 0BFF.

Un programme qui est mis au point, peut être figé en EPROM, EPROM qui est ensuite placée sur le support destiné à cet effet de la *maquette processeur.*

I. 2. Le mode « mise au point d'un ensemble autonome à microprocesseur » MODE II

Ce mode comporte deux étapes :

a) *Etape 1 :*

L'utilisateur conçoit et réalise une maquette dite « maquette application » et la remplace par la maquette « entrée/sortie ». Cette « maquette application » correspond à une application quelconque souhaitée par l'utilisateur (par exemple : horloge, synthétiseur, thermomètre, etc.). L'utilisateur met au point le programme répondant à la fonction qu'il souhaite ; ce programme est dans la RAM 2 K octets.

b) *Etape 2 :*

L'utilisateur place une EPROM 2716 vierge sur le support de la maquette VILÉMIO 2 destiné à cet effet, et transfère dans celle-ci le programme actuellement placé dans la RAM 2 K : une EPROM APPLICATION est ainsi fabriquée.

Cette EPROM APPLICATION est ensuite plaçée sur le support de la maquette VILÉMIO 1, puis VILÉMIO 1 est directement connectée à la maquette application (l'ensemble VILÉMIO 2/3 est supprimé) : l'utilisateur dispose ainsi d'un système autonome à microprocesseur répondant à une fonction déterminée.

II. Précautions à prendre lors du transfert d'une zone mémoire de RAM en EPROM

Avant son transfert en EPROM, un programme qui vient d'être mis au point est dans la RAM située aux adresses de 0400 à 0BFF. Appelons n_{RAM} l'adresse à laquelle débute ce programme.

Dans le MODE 1, ce programme est transféré dans une EPROM qui doit être située aux adresses de *1000 à 17FF.*

Dans le MODE II, ce programme est transféré dans une EPROM qui sera située (après déconnexion de l'ensemble VILÉMIO 2/3) aux adresses de *0000 à 07FF.*

Appelons n_{EPR} l'adresse à laquelle débute ce programme, *mais dans l'EPROM.*

Avant le transfert en EPROM, il y a des précautions à prendre concernant l'adressage de la mémoire. Deux cas sont à envisager :

1. *Les 8 bits de poids faible de n_{RAM} sont identiques aux 8 bits de poids faible de n_{EPR} :*

par exemple : $n_{RAM} = 0500$ et $n_{EPR} = 1000$ (MODE I).

Si le programme contient des instructions de *branchement long* (de code C O, C 1, C 2, C 3, C 9, C A et C B), leur code est associé à une adresse XYZT (deux octets) qui doit être modifiée, s'il s'agit d'une adresse de branchement à l'intérieur du programme qu'on désire transférer en EPROM. Cette modification transforme l'adresse XYZT en l'adresse X'Y'Z'T' selon la relation suivante :

$$(X'Y'Z'T') = (XYZT) + (n_{EPR} - n_{RAM})$$

Par exemple, à l'adresse 0610 de la RAM, il y a une instruction de branchement long telle que (XYZT) = 0720. Si $n_{ram} = 0500$ et $n_{EPR} = 1\,000$ (MODE I), l'adresse (XYZT) doit être transformée en l'adresse (X'Y'Z'T') égale à $(X'Y'Z'T') = 0720 + (1000 - 0500) = 1220$.

2. *Les 8 bits de poids faible de n_{RAM} sont différents des 8 bits de poids faible de n_{EPR} :*

par exemple $n_{RAM} = 04F0$ et $n_{EPR} = 0000$ (MODE II).

Si le programme contient des instructions de branchement long, il faut modifier les adresses de la même façon qu'indiqué plus haut.

Mais il faut également modifier les adresses des *branchements courts*. Pour éviter le fastidieux travail de calcul des nouvelles adresses, il suffit de *translater* le programme situé en RAM vers une zone pour laquelle les 8 bits de poids faible de n_{RAM} sont identiques aux 8 bits de poids faible de n_{EPR} ; les nouvelles adresses (concernant les branchements courts) sont automatiquement calculées par la routine incluse dans l'EPROM MONITEUR.

Cette troisième partie ne propose pas la réalisation pratique effective de « maquettes application » (faute de place et de temps), mais propose des idées que nous avons développées en utilisant le système VILÉMIO dans le MODE 1. Au lecteur de passer au MODE II, c'est-à-dire d'étendre ces idées, de mettre au point lui-même des applications personnelles.

Troisième partie

Quelques idées pour vos projets à microprocesseur

Chapitre 12

LA COMMANDE D'UN TRIAC

I. Commande de l'angle d'amorçage d'un triac

I. 1. Commande d'un triac

Un montage de principe de commande impulsionnelle de triac est donné figure XII. 1. Le transformateur permet d'isoler le générateur d'impulsions du réseau alternatif 220 V ; la circuiterie destinée à protéger le triac lors de la commutation sur charge inductive n'est pas représentée.

La figure XII. 2 indique, dans le cas d'une charge résistive (ou pratiquement résistive comme une ampoule électrique par exemple), la forme de la tension u_c aux bornes de la charge lorsque la gachette G est attaquée par des impulsions d'amplitude et de largeur suffisantes. Sur l'axe horizontal sont représentés non pas les temps, mais les arguments ; ainsi, à une demi-période $T/2 = 10$ ms (sauf pour nos lecteurs d'outre-atlantique où celle-ci vaut 8,33.. ms) correspond un argument de π radians ou 180 degrés. L'angle θ est appelé *angle d'amorçage*. La valeur efficace U_{Ceff} de la tension aux bornes de la charge dépend de θ : si $\theta = 0$, U_{Ceff} est maximale et vaut 220 V ; si $\theta = 180^o$, U_{Ceff} est minimale et vaut 0 V. Ainsi, la variation de l'angle d'amorçage θ provoque la variation de la puissance dissipée dans la charge, c'est-à-dire la variation de la luminosité de la lampe s'il s'agit d'une lampe.

Pour nos essais personnels, nous avons en fait utilisé ce qu'on appelle un relais statique contenant dans un seul boîtier 14 broches « dual-in-line » le triac, le transformateur d'isolement, et la circuiterie de contrôle de la gachette. Nous avons utilisé un relais statique type 641-2 TELEDYNE. La figure XII. 3 indique un schéma de commande possible ; la base du transistor est activée, par l'intermédiaire d'une résistance R_B de 4,7 kΩ, par la sortie Q du microprocesseur : lorsque Q est au niveau logique 1, le transistor 2N22 22 est saturé et le triac peut conduire ; lorsque Q est au niveau 0, le transistor est bloqué et le triac se bloque lors du passage par zéro de la tension du réseau.

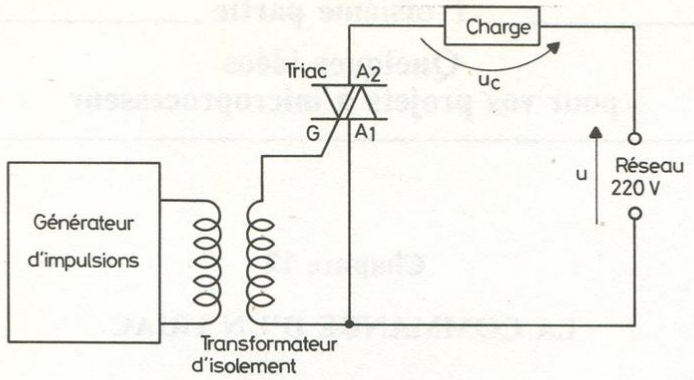


Fig. XII. 1. Principe de la commande de triac

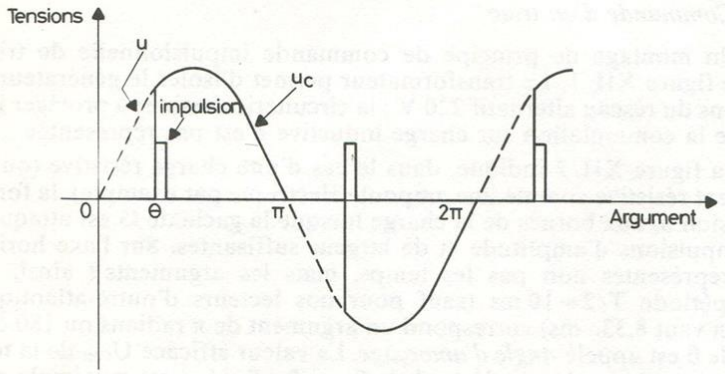


Fig. XII. 2. Forme des tensions

I. 2. Commande par microprocesseur

Pourquoi ne pas utiliser un microprocesseur pour commander le triac ? Même si cela peut paraître plus compliqué que les montages traditionnels, nous pensons qu'on a toujours intérêt à transposer les techniques classiques dans le domaine des microprocesseurs, à cause du gain en souplesse et en finesse qui en résulte. Par ailleurs, on augmente son potentiel d'expérience quant aux problèmes d'interfaçage entre le microprocesseur et le monde extérieur.

La première chose à voir est qu'il s'agit de générer une impulsion sur $Q,\theta/\pi$ demi-période après chaque passage par zéro de la tension u fournie par le réseau : il faut donc que le microprocesseur « reconnaisse » ces passages

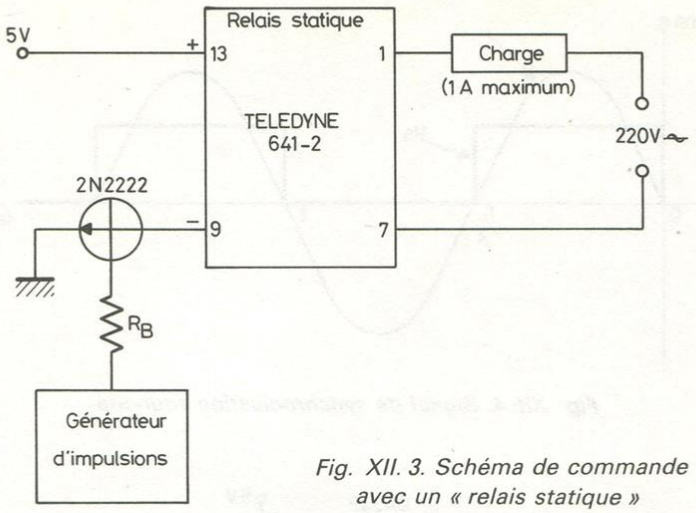


Fig. XII. 3. Schéma de commande avec un « relais statique »

par zéro. Autrement dit, il faut que le générateur d'impulsions (en fin de compte le microprocesseur lui-même) soit *synchronisé* sur le réseau.

La deuxième chose à voir est que l'angle d'amorçage θ est nécessairement *numérisé* (quantifié) dans le microprocesseur. Il faut donc définir une *résolution* pour θ ; nous avons pensé qu'il était raisonnable de pouvoir faire varier θ de degré en degré, ce qui nécessite 8 bits puisque $127 < 180 < 255$. Autrement dit, c'est un mot W de longueur maximale 8 bits, situé en mémoire, qui définira l'angle θ.

I. 3. La synchronisation sur le réseau

Pour que le microprocesseur reconnaisse les passages par zéro de la tension u fournie par le réseau, il faut générer un signal u_s (que nous appelons signal de synchronisation) variant de 0 à 5 V en synchronisme avec le réseau. La figure XII. 4 indique le but souhaité, et la figure XII. 5 un moyen de l'atteindre. Le CA 339 (RCA) est un comparateur qui, alimenté sous 5 V, fournit une tension u_s égale à 5 V si e_D est positive, et égale à 0 V si la tension e_D est négative. Les résistances R_1 et R_2 forment un pont diviseur destiné à protéger le comparateur. En effet, par rapport à la masse, les tensions sur les entrées (+) et (−) du comparateur n'ont pas le droit d'être inférieures à -300 mV ; par conséquent, pour une tension sinusoïdale v dont la valeur efficace V est 12 V par exemple, on peut prendre $R_1 = 22\ k\Omega$ et $R_2 = 220\ \Omega$, ce qui donne

$$|e_D| \leqslant \frac{R_2}{R_1 + R_2} V \cdot \sqrt{2} = 170\ \text{mV} < 300\ \text{mV}.$$

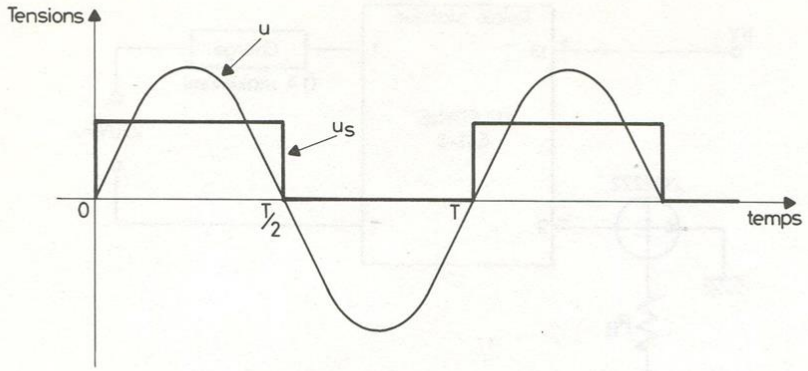


Fig. XII. 4. Signal de synchronisation souhaité

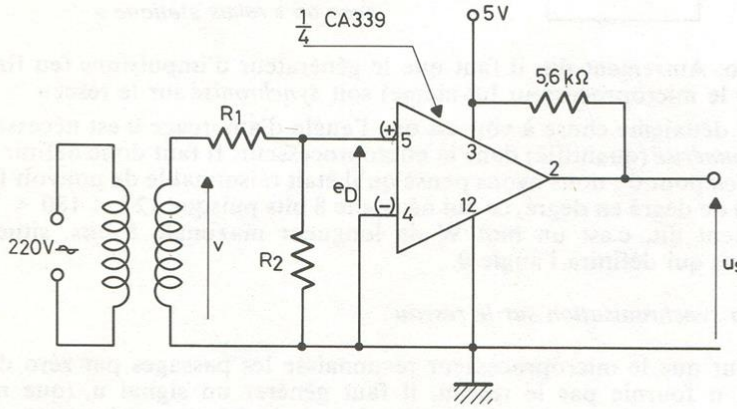


Fig. XII. 5. Circuit d'obtension du signal de synchronisation

Le transformateur, qui peut être d'un type quelconque et de très faible puissance, ne sert qu'à l'isolement du montage par rapport au réseau ; il n'est pas strictement nécessaire mais permet d'avoir la masse du montage flottante et isolée.

Le signal logique u_s doit être introduit dans le microprocesseur. Que faut-il faire ? Nous avons déjà dit que $\Delta = \theta/\pi \times T/2$ secondes après chaque front montant ou descendant le u_s, il fallait générer une impulsion sur la sortie Q ; c'est ce que représente l'organigramme général de la figure XII.6. Il apparaît ainsi qu'il faut que le microprocesseur détecte une *variation* (un front) de la tension u_s.

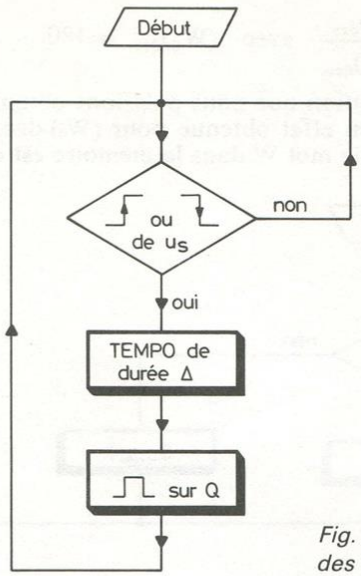


Fig. XII. 6. Organigramme de la génération des impulsions de sortie

I. 4. Une méthode de détection d'un front

Détecter un front, cela veut dire fabriquer une branche d'un organigramme dans laquelle le microprocesseur ne « passe » qu'une fois au moment du front. Cela pose un petit problème car les entrées du microprocesseur sont sensibles à des niveaux et non à des fronts ; c'est le cas en particulier de l'entrée $\overline{EF_1}$ que nous avons choisie et donc reliée à la sortie du comparateur. Ce problème est résolu (de façon logicielle), comme le montre la figure XII. 7, par l'intervention d'un indicateur I interne au microprocesseur. Après un instant d'attention, le lecteur s'apercevra que les branches notées ⊣↓ et ⊣↑ ne sont « parcourues » qu'au moment de l'apparition de ces fronts. Pour l'indicateur I, nous avons pris le demi-registre R (4).0, pourquoi pas ?

I. 5. Le réglage de l'angle d'amorçage

Soit un mot W de 8 bits situé dans la mémoire à l'adresse n pointée par le registre R (2). (fig. XII. 8). Nous supposons que ce mot W est placé dans la mémoire par l'utilisateur à l'aide du moniteur, ou qu'il y est parce que le microprocesseur a exécuté une instruction INPUT par exemple.

Lorsqu'un front de u_s est détecté, le microprocesseur doit décrire une boucle de temporisation de durée Δ. Nous décidons *a priori* de rendre cette durée Δ proportionnelle à W selon l'expression suivante :

$$\frac{2}{T} \times \Delta = \frac{\theta\ (\text{deg.})}{180°} = \frac{(W)_{dec.}}{(W_0)_{dec.}} \quad \text{avec} \quad (W_0)_{dec.} = 180.$$

Ce n'est pas la plus haute résolution que nous puissions obtenir avec un mot W de 8 bits, celle-ci étant en effet obtenue pour (W_0) déc. = 255. Cependant notre choix entraîne que le mot W dans la mémoire est exacte-

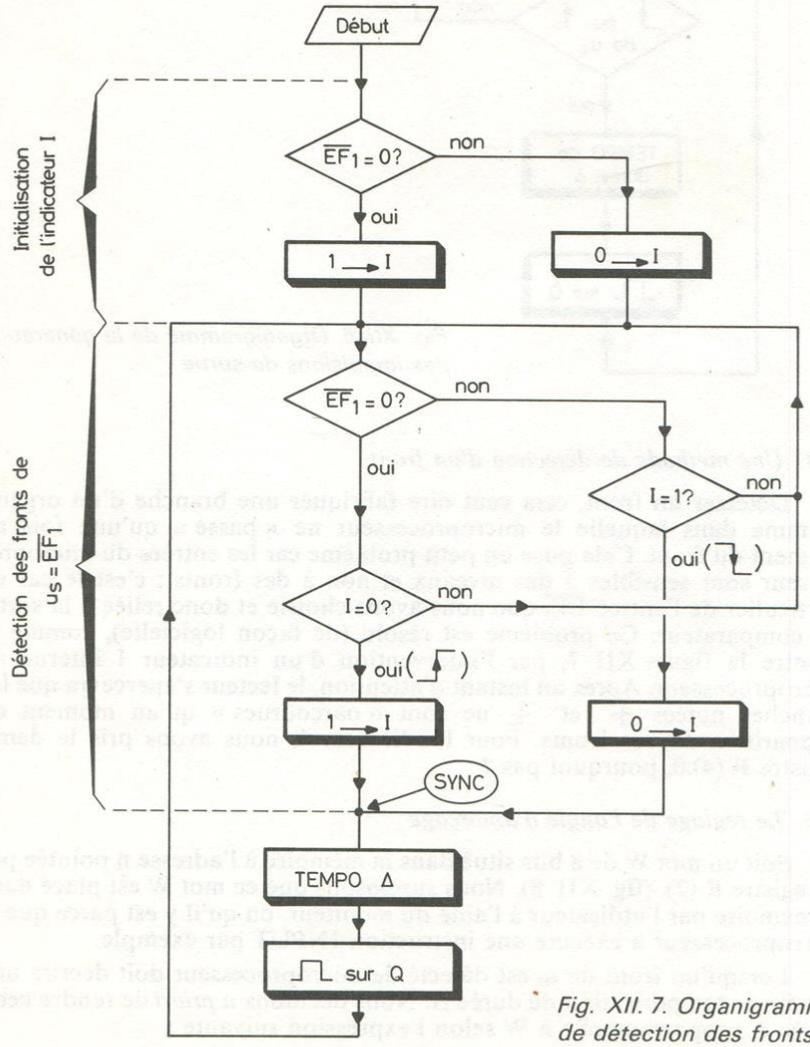


Fig. XII. 7. Organigramme de détection des fronts

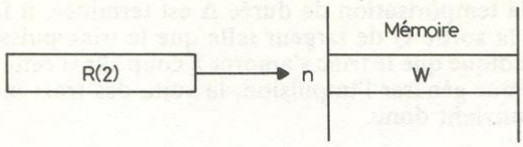


Fig. XII. 8. Emplacement du mot W

ment la représentation binaire de l'angle d'amorçage θ en degrés. A 180° correspond une demi-période du réseau de 10 ms pour nos lecteurs européens, soit 2500 cycles machine pour une fréquence du quartz de 2 MHz (1 cycle machine = 4 μs) ; donc à 1° correspond

$$\frac{2\,500}{180} = 13{,}89$$

cycles machine, ce qui n'est pas un chiffre rond. Comme le microprocesseur ne peut décrire des fractions de cycle, nous arrondissons ce chiffre à 14 ce qui nous amène à modifier la résolution choisie en prenant $(W_0)_{dec.}$ de telle sorte que la relation $\frac{2\,500}{(W_0)_{dec.}} = 14$ soit vérifiée ; on trouve $(W_0)_{dec.} = 178{,}57$, ce qui entraîne $(W)_{dec.} = 0{,}99.\ \theta\ (deg.)$: le mot W dans la mémoire est donc la représentation binaire de l'angle d'amorçage θ en degrés, à *1 % près.*

Il en résulte le tableau XII. 1 qui indique, en langage assembleur, la structure du bloc de temporisation Δ ; on suppose que le mot W a été transféré dans l'accumulateur D. Pour une fréquence d'horloge de 2 MHz, la durée nécessaire à l'exécution de ce bloc de temporisation est de 14. $(W)_{dec.}$ cycles machine.

Adresse	Mnémonique	Nombre de cycles	Commentaires
	Instructions précédentes		D contient W
TEMP	NOP	3	Instructions sans effet
	NOP	3	
	SEX R (N)	2	
	SEX R (N)	2	
	SMI 01	2	D—01→D
	BNZ TEMP	2	
FIN TEMP	Instructions suivantes		14 x $(W)_{déc}$ cycles ont été décrits

Tableau XII. 1. Structure du bloc de temporisation

Lorsque la temporisation de durée Δ est terminée, il faut générer une impulsion sur la sortie Q de largeur telle que le triac puisse s'amorcer. Le constructeur indique que le triac s'amorce à coup sûr si cette largeur est d'au moins 20 µs. Pour générer l'impulsion, la suite des trois instructions SEQ, NOP, REQ, convient donc.

I. 6. Mise en œuvre et programme

a) *Utilisation de l'instruction INPUT*

Pour un essai rapide, on peut décider d'utiliser le registre d'entrée de la maquette Entrée/Sortie pour transférer le mot W dans la mémoire. Ce mot W doit donc être présenté par l'utilisateur sur les cosses marquées DI_7 à DI_0.

Nous décidons de demander au microprocesseur de scruter le registre d'entrée à chaque front de la grandeur u_s, par le moyen de l'instruction INPUT de code 69. Cette instruction transfère le contenu du registre d'entrée dans la mémoire, à l'adresse définie par le contenu du registre R (X). Comme l'indique la figure XII. 9, l'instruction INPUT est exécutée

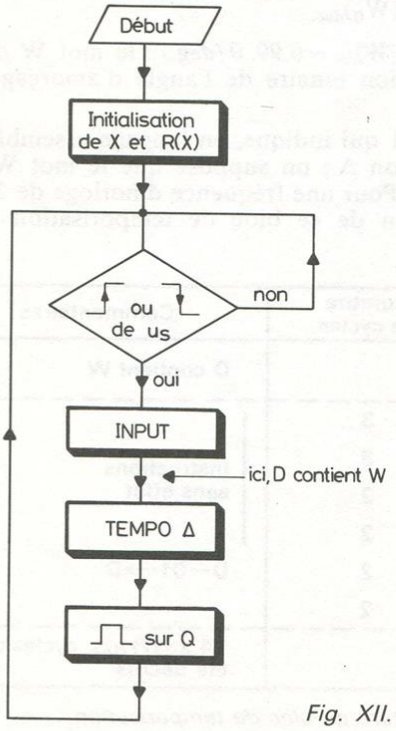


Fig. XII. 9. Utilisation de l'instruction INPUT

toutes les 10 ms, ce qui fait que le mot W présenté en entrée du registre est « instantanément » pris en compte.

b) *le programme de commande de l'angle d'amorçage*

est donné au tableau XII. 2. Ce programme commence à l'adresse 0400, et le registre R (2) est initialisé à 0C10 (pourquoi pas ?).

Langage machine		Langage assembleur		
Adresse	Code et opérande	Adresse	Mnémonique	Commentaires
0400	E2		SEX R (2)	
1	F8 0C		LDI 0C	R (2) initialisé
3	B2		PHI R (2)	à 0C10
4	F8 10		LDI 10	et X vaut 2
6	A2		PLO R (2)	
7	F8 01		LDI 01	
9	3C 0C		BN1 DEDE	
B	38		SKP	Initialisation
C	24	DEDE	DEC R (4)	de l'indicateur I
D	A4		PLO R (4)	(I = R (4). 0)
E	3C 15	ALO	BN1 KLA	
0410	84		GLO R (4)	
1	32 29		BZ MON	
3	30 0E		BR ALO	
5	84	KLA	GLO R (4)	Détection des
6	32 0E		BZ ALO	fronts de U_s
8	F8 00		LDI 00	front
A	A4		PLO R (4)	0→ I
B	69		INP R (2)	Entrée de W
C	C4	TEMP	NOP	
D	C4		NOP	
E	E2		SEX R (2)	Tempo durée Δ
F	E2		SEX R (2)	
20	FF 01		SMI 01	
2	3A 1C		BNZ TEMP	
4	7B		SEQ	
5	C4	IMP	NOP	Impulsion sur Q
6	7A		REQ	
7	30 0E		BR ALO	
9	F8 01	MON	LDI 01	front
B	A4		PLO R (4)	1→ I
042C	30 1C		BR TEMP	

Tableau XII. 2. Programme de commande de l'angle d'amorçage

c) *Mise en œuvre*

On peut réaliser le montage comprenant le comparateur, le relais TELEDYNE et les divers autres composants sur une plaquette séparée. Les liaisons entre cette plaquette et le système VILÉMIO sont réduites (outre l'alimentation 5 V) à deux cordons :

— la sortie Q qui commande le relais statique ;

— l'entrée $\overline{EF_1}$ qui est à relier à la sortie du comparateur. Remarquez que $\overline{EF_1}$ ne doit être relié à cette sortie qu'après écriture du programme dans la RAM et lancement de celui-ci. En effet, l'entrée $\overline{EF_1}$ est aussi utilisée par le programme moniteur lors de la scrutation du clavier.

Pour nos essais, notre charge était une ampoule électrique de 100 W (le relais choisi peut commuter 110 W sans radiateur, et 220 W avec radiateur) ; le programme lancé, on observe une variation continue de la puissance lumineuse lorsque la donnée DI présentée sur les entrées DI_7 à DI_0 varie.

II. Synchronisation par interruption de programme

II. 1. Génération des interruptions

Pour synchroniser le microprocesseur sur le réseau, on peut fort bien utiliser son entrée $\overline{INT}$ en s'arrangeant pour l'activer (la mettre au niveau logique 0) à chaque front montant *et* descendant de u_s. Comme l'entrée $\overline{INT}$ est sensible à un *niveau* (et non à un front), il est nécessaire de la commander par l'intermédiaire d'une bascule remise automatiquement à zéro lorsque la requête d'interruption est acquiescée. On aboutit au montage de la figure XII. 10 qui demande quelques commentaires :

— observez qu'à cause du branchement des entrées du comparateur, les grandeurs u_s et $\overline{u_s}$ sont complémentaires. L'existence du signal $\overline{u_s}$ est justifiée par le fait qu'il est nécessaire de disposer d'un front *montant* toutes les 10 ms (alors que dans le montage précédent, on disposait alternativement d'un front montant puis descendant) ; en effet, les entrées Cl_1 et Cl_2 de la double bascule JK CD 4027 (RCA) sont sensibles au front montant ;

— les chronogrammes de la figure XII. 11 donnent l'évolution des signaux utiles. Prenons l'entrée Cl_1 par exemple : un front montant sur Cl_1 entraîne la mise au niveau logique 0 de $\overline{Q_1}$, établissant ainsi une requête d'interruption ; le microprocesseur termine alors l'exécution de l'instruction en cours et s'engage dans un cycle d'interruption, ce qu'il signale en particulier en envoyant un niveau logique 1 sur la ligne SC1. Comme SC1 est relié à l'entrée R_1 (remise à zéro) de la bascule, la sortie $\overline{Q_1}$ revient au niveau 1 de telle sorte qu'elle soit prête à activer de nouveau l'entrée $\overline{INT}$ du microprocesseur lors de la requête suivante d'interruption.

Ainsi, une interruption et une seule est générée exactement toutes les 10 ms.

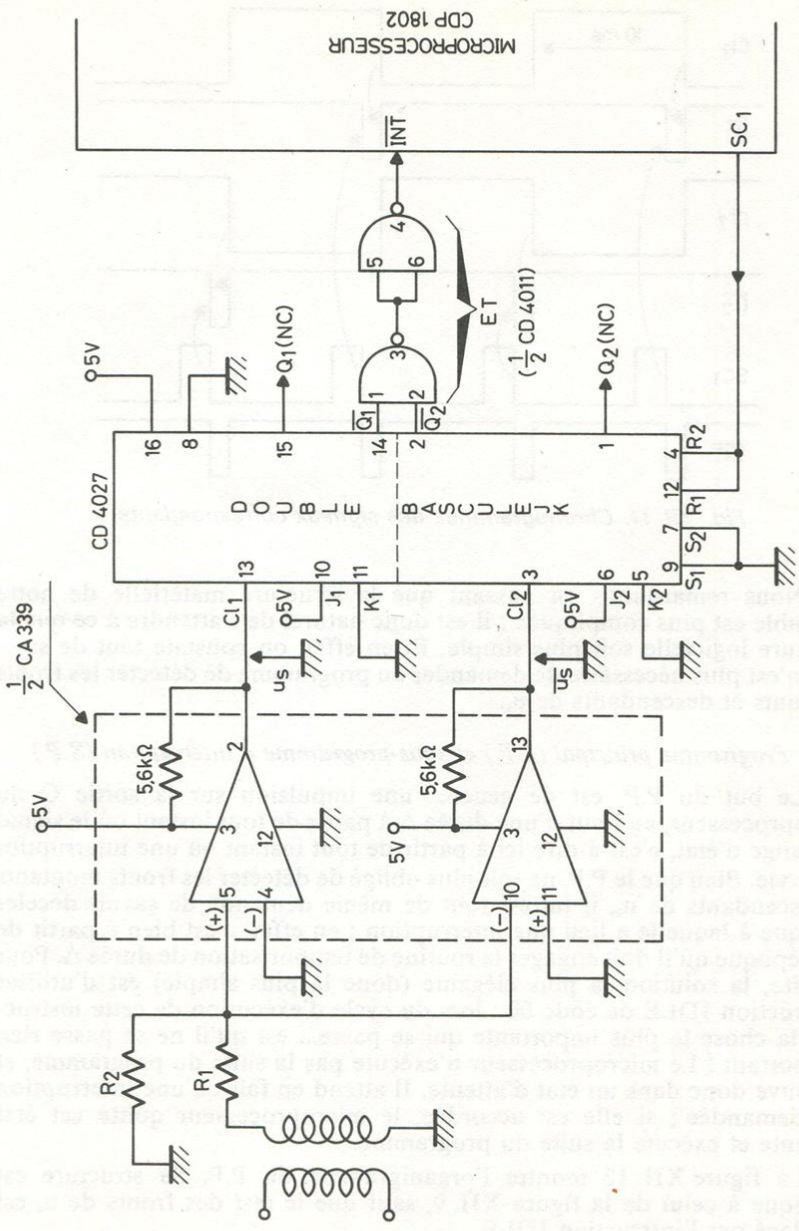


Fig. XII. 10. Génération des interruptions

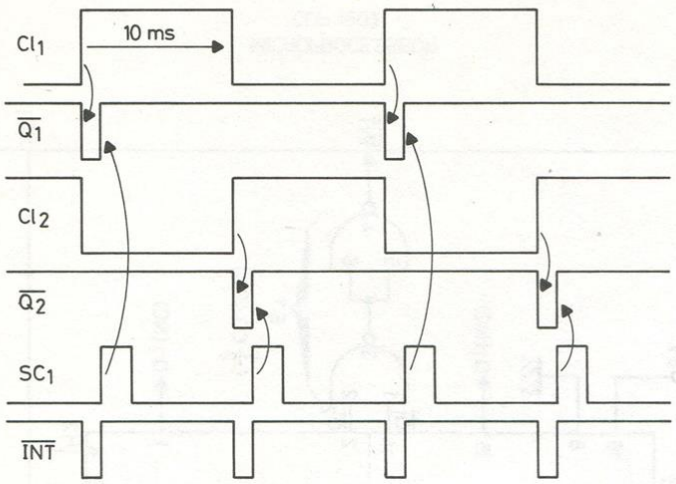


Fig. XII. 11. Chronogrammes des signaux correspondants

Nous remarquons en passant que la structure matérielle de notre ensemble est plus compliquée ; il est donc naturel de s'attendre à ce que la structure logicielle soit plus simple. Et en effet, on constate tout de suite qu'il n'est plus nécessaire de demander au programme de détecter les fronts montants et descendants de u_s.

II. 2. Programme principal (P.P.) et sous-programme d'interruption (S.P.)

Le but du P.P. est de générer une impulsion sur la sortie Q du microprocesseur, au bout d'une durée Δ à partir de tout instant où le signal u_s change d'état, c'est-à-dire ici à partir de tout instant où une interruption est servie. Bien que le P.P. ne soit plus obligé de détecter les fronts montants et descendants de u_s, il lui est tout de même demandé de savoir déceler l'époque à laquelle a lieu une interruption ; en effet, c'est bien à partir de cette époque qu'il doit engager la routine de temporisation de durée Δ. Pour ce faire, la solution la plus élégante (donc la plus simple) est d'utiliser l'instruction IDLE de code 00 : lors du cycle d'exécution de cette instruction, la chose la plus importante qui se passe... est qu'il ne se passe rien d'important ! Le microprocesseur n'exécute pas la suite du programme, et se trouve donc dans un état d'attente. Il attend en fait qu'une interruption soit demandée ; si elle est accordée, le microprocesseur quitte cet état d'attente et exécute la suite du programme.

La figure XII. 12 montre l'organigramme du P.P. Sa structure est identique à celui de la figure XII. 9, sauf que le test des fronts de u_s est remplacé par l'instruction IDLE.

Les initialisations concernent :

— le registre R (1) qui est le compteur de programme du sous-programme d'interruption ;

— le pointeur de pile R (2), destiné d'une part à pointer la ligne mémoire dans laquelle va être stockée la donnée W introduite lors de l'exécution de INPUT, et d'autre part à pointer la ligne mémoire dans laquelle va être stocké le couple (X, P) au cours de la routine d'interruption.

La figure XII. 13 montre l'organigramme du S.P. Le rôle de ce S.P. est de ne rien faire qu'exister ! On lui demande cependant d'être capable de revenir au P.P. et de restituer à X sa valeur primitive. C'est pourquoi on sauvegarde l'ensemble (X, P) du P.P. dans la pile (avec l'instruction SAVE), puis on restitue l'ensemble (X, P) au P.P. (par l'instruction RETURN).

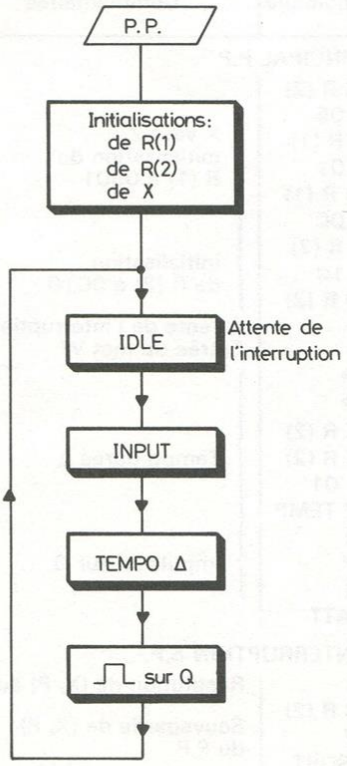


Fig. XII. 12. Organigramme du programme principal

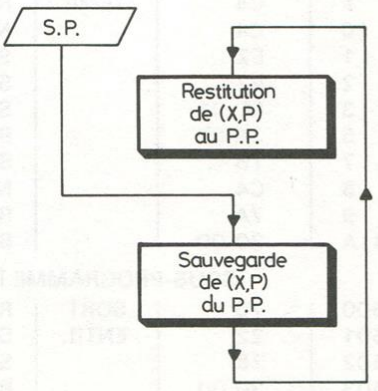


Fig. XII. 13. Organigramme du sous-programme d'interruption

II. 3. Le programme est présenté au tableau XII. 3

Le P.P. commence à l'adresse 0400 et le S.P. à l'adresse 0501. Il est à noter que l'entrée $\overline{INT}$ du microprocesseur ne doit être reliée au montage qu'après lancement du programme ; ensuite il n'y a pas de précautions à prendre lorsqu'on revient au moniteur (l'entrée $\overline{INT}$ peut rester connectée au générateur d'interruptions) puisque R (1) est défini.

La sortie SC1 est disponible sur la maquette processeur.

Adresse	Code et opérande	Adresse	Mnémonique	Commentaires
		PROGRAMME PRINCIPAL P.P		
0400	E2		SEX R (2)	
1	F8 05		LDI 05	X vaut 2 initialisation de R (1) à 0501
3	B1		PHI R (1)	
4	F8 01		LDI 01	
6	A1		PLO R (1)	
7	F8 0C		LDI 0C	Initialisation de R (2) à 0C10
9	B2		PHI R (2)	
A	F8 10		LDI 10	
C	A2		PLO R (2)	
D	00	ATT	IDL	Attente de l'interruption
E	69		INP	Entrée du mot W
F	C4	TEMP	NOP	Tempo durée Δ
10	C4		NOP	
1	E2		SEX R (2)	
2	E2		SEX R (2)	
3	FF 01		SMI 01	
5	3A 0F		BNZ TEMP	
7	7B		SEQ	Impulsion sur Q
8	C4		NOP	
9	7A		REQ	
041A	30 0D		BR ATT	
		SOUS-PROGRAMME D'INTERRUPTION S.P.		
0500	70	SORT	RET	Restitution de (X, P) au P.P
0501	22	ENTR.	DEC R (2)	Sauvegarde de (X, P) du P.P
0502	78		SAV	
0503	30 00		BR SORT	

Tableau XII. 3. Programme principal et sous-programme d'interruption

III. Remarques

III. 1. Utilisation de l'entrée $\overline{\text{DMA-OUT}}$ du microprocesseur

Il faut rappeler que l'instruction IDLE met le microprocesseur dans un état d'attente d'une interruption de programme, qu'elle vienne par le canal INTERRUPT *ou le canal DMA.*

Autrement dit, si nous générons des interruptions sur l'entrée $\overline{\text{DMA-OUT}}$ (et non plus l'entrée $\overline{\text{INT}}$), le résultat sera exactement le même puisque le sous-programme d'interruption n'a aucune tâche à accomplir. L'avantage de ce procédé est que le programme qui en résulte est nettement plus simple ; en effet, il n'est plus nécessaire d'initialiser le registre R (1), ni bien sûr d'écrire de sous-programme d'interruption.

Dans ce montage de commande de triac, l'opération effectuée lors de l'exécution du cycle DMA (c'est-à-dire M (R(0)) → Bus des données ; R(0) + 1 → R(0)) n'est pas utilisée. Seul est utilisé le fait que l'activation de $\overline{\text{DMA-OUT}}$ *déclenche* l'exécution du programme à partir de l'instruction IDLE.

III. 2. Autre application d'une base de temps synchronisée sur le réseau

Reprenez par exemple l'organigramme de la figure XII. 7. Il n'est pas douteux qu'on dispose d'une base de temps synchronisée sur le réseau, puisque le programme passe au point marqué SYNC à chaque front de u_s.

On peut s'arranger facilement pour passer en ce point seulement toutes les 0,5 seconde par exemple (en utilisant un registre initialisé à la valeur $(50)_{dec} = (32)_{hex}$). Si, en outre, on remplace l'ensemble « TEMPO Δ et IMPULSION sur Q » par l'ensemble « Q change d'état », le triac commande successivement l'allumage et l'extinction de l'ampoule électrique exactement au rythme de 1 Hz.

Naturellement, on peut utiliser cette base de temps pour concevoir tout jeu de lumière ou toute horlogerie aussi complexe que l'on voudra. Dans le cadre de cet ouvrage, nous ne pouvons (ni ne voulons) développer une application complète de ce genre : nous ne le pouvons faute de place et de temps ; nous ne le voulons car chacun d'entre nous a ses propres centres d'intérêt et buts particuliers. Nous cherchons seulement à donner quelques idées de base, dont la combinaison et l'exploitation permettent d'aboutir à la conception d'un projet que vous avez vous-même défini.

Chapitre 13

LA COMMANDE D'UN MOTEUR PAS A PAS

I. Principe d'un moteur pas à pas

Un moteur pas-à-pas est un moteur qui, à chaque impulsion de commande, voit sa rotation progresser d'un angle appelé *angle de pas*. Nous ne nous intéressons ici qu'aux moteurs dont le rotor est un aimant permanent.

Les figures XIII. 1-a, 1-b, 1-c, 1-d montrent les quatre états d'excitation possibles pour un moteur pas à pas classique à deux stators. Le stator 1, correspondant aux deux demi-enroulements P et Q, fabrique dans son entrefer le champ magnétique B_1. Le stator 2, correspondant aux deux demi-enroulements R et S, fabrique dans son entrefer le champ magnétique B_2. Le champ magnétique résultant est noté B. Un moteur possédant quatre demi-enroulements est dit à quatre *phases*. Par ailleurs, comme le point milieu des enroulements est porté, par rapport à la masse, à un potentiel (égal à E) de polarité unique, un tel moteur est dit *unipolaire*.

Les figures XIII. 1 montrent que selon les quatre positions indiquées des commutateurs électroniques S_1 et S_2, le champ magnétique B est orienté selon quatre positions différentes, décalées entre elles de 90°. Si un aimant permanent constitué par une paire de pôles (un pôle Nord et un pôle Sud) est placé dans l'entrefer, il est soumis à un couple qui tend à l'orienter dans le sens de B ; cet aimant constitue le rotor du moteur, et tourne d'un quart de tour à chaque commutation : on dispose alors d'un moteur pas à pas d'angle de pas égal à 90°.

Pour diminuer l'angle de pas (ou augmenter le nombre de pas par tour), on peut augmenter le nombre de phases (ce qui complique la commande) ou bien augmenter le nombre de paires de pôles du rotor : pour un moteur *4* phases dont le rotor présente *12* paires de pôles, l'angle de pas est de

$\frac{90°}{12} = 7°30$, ce qui donne 48 pas par tour.

Le principe de fonctionnement des figures XIII. 1 permet de déterminer les chronogrammes de la figure XIII. 2, qui indiquent comment alimenter les quatre phases pour faire tourner le moteur pas à pas, dans l'un ou l'autre sens ; le niveau haut sur une phase veut dire : phase alimentée, le niveau bas : phase non alimentée. La commande d'un moteur pas à pas de

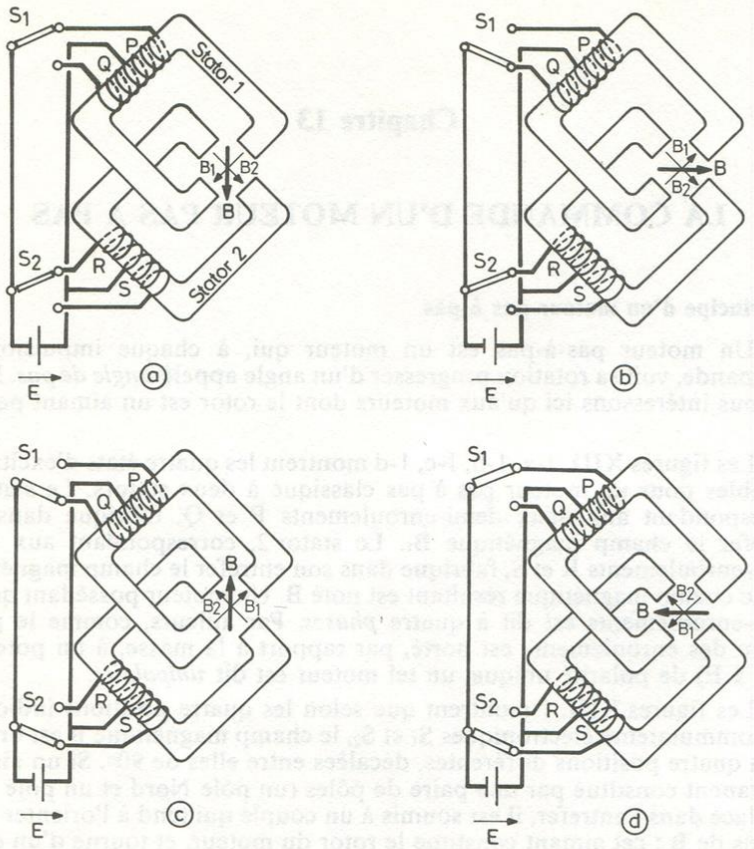


Fig. XIII. 1. Principe d'un moteur pas à pas

quatre phases pose donc au moins le problème de la génération de signaux, selon les séquences de la figure XIII. 2.

II. Commande d'un moteur pas à pas. Variante I

II. 1. Le dispositif de puissance

Disons tout de suite que pour nos essais personnels, nous avons utilisé un moteur pas à pas unipolaire quatre phases à aimant permanent de chez CROUZET type 827904, à angle de pas de 7°30 (48 pas par tour).

Le premier circuit que nous avons réalisé est présenté en figure XIII. 3.

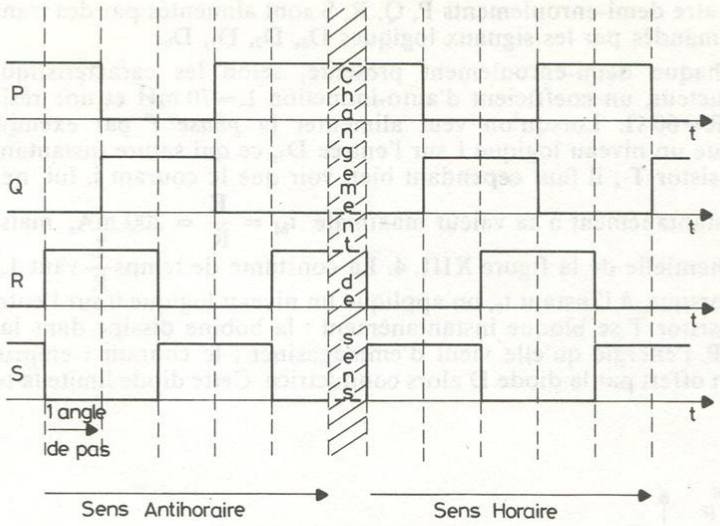


Fig. XIII. 2. Chronogrammes d'alimentation

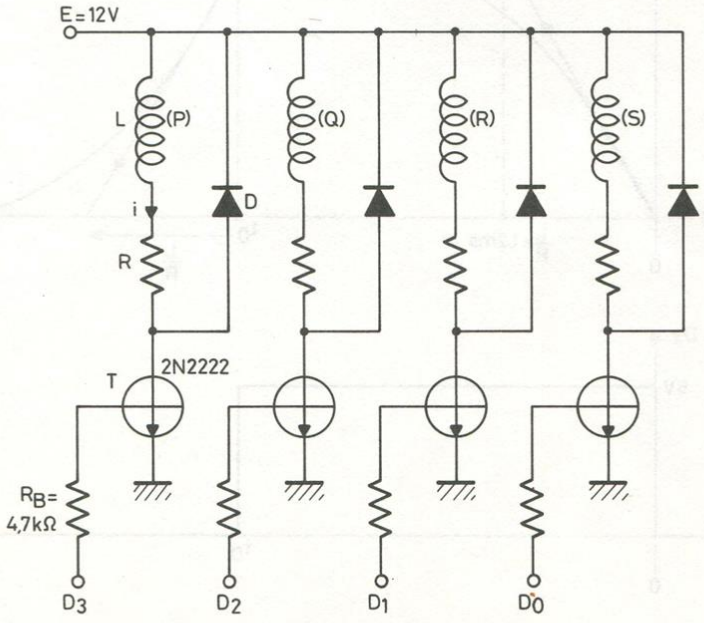


Fig. XIII. 3. 1er circuit de commande

Les quatre demi-enroulements P, Q, R, S sont alimentés par des transistors T commandés par les signaux logiques D_3, D_2, D_1, D_0.

Chaque demi-enroulement présente, selon les caractéristiques du constructeur, un coefficient d'auto-induction $L = 70$ mH et une résistance série $R = 60\ \Omega$. Lorsqu'on veut alimenter la *phase P* par exemple, on applique un niveau logique 1 sur l'entrée D_3, ce qui sature instantanément le transistor T ; il faut cependant bien voir que le courant i, lui, ne passe pas instantanément à sa valeur maximale $i_M = \frac{E}{R} = 200$ mA, mais selon l'exponentielle de la figure XIII. 4. La constante de temps $\frac{L}{R}$ vaut 1,2 ms.

Lorsque, à l'instant t_0, on applique un niveau logique 0 sur l'entrée D_3, le transistor T se bloque instantanément : la bobine dissipe dans la résistance R l'énergie qu'elle vient d'emmagasiner ; le courant i emprunte le chemin offert par la diode D alors conductrice. Cette diode limite la tension

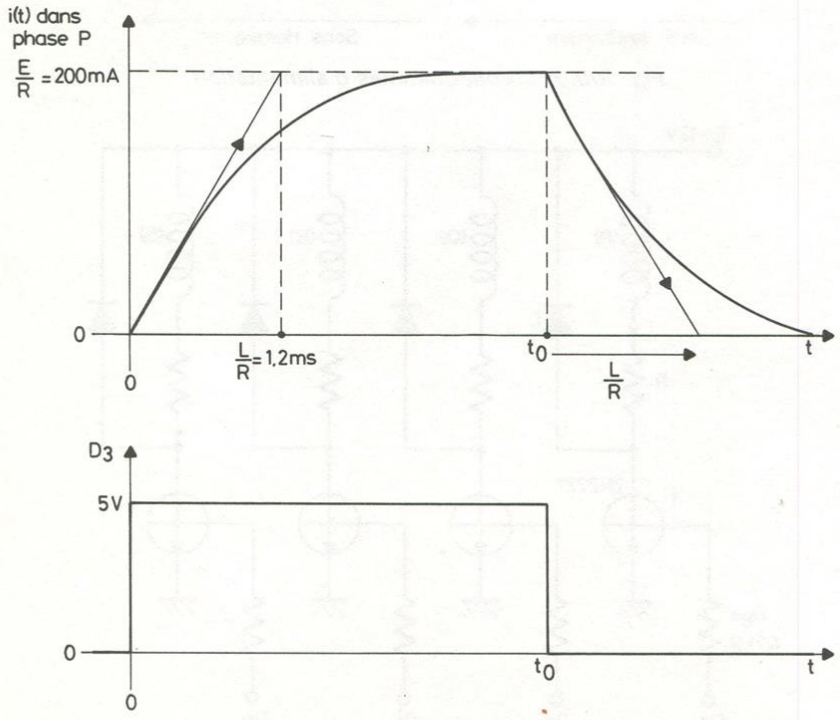


Fig. XIII. 4. Allure des courants en fonction des tensions de commande

V_{CE} entre collecteur et émetteur du transistor, à la valeur $E + V_D$ soit environ 12,8 V (V_D est la tension de seuil de la diode). Si on ne met pas de diode, il apparaît une surtension aux bornes du transistor, qui a pour effet de placer celui-ci dans sa zone d'avalanche ; si le transistor n'est pas prévu pour supporter cela (c'est le cas du 2N2222)... il ne le supporte pas !

Ce que nous venons de dire montre que les caractéristiques électriques du moteur pas à pas, imposent en elles-mêmes une limitation dans sa vitesse de rotation. En effet, voyez la figure XIII. 5 où on prétend commander le moteur avec un signal D_3 tel que t_0 soit petit devant $\frac{L}{R}$ (fréquence de plusieurs kilohertz) : le courant i dans les phases varie très peu, d'où il ne résulte plus aucun effet mécanique sur l'arbre du moteur.

Remarque :

Des montages électroniques plus élaborés utilisant en particulier un réseau de compensation R, C permettent cependant de reculer la limite supérieure de la vitesse de rotation, en améliorant le temps de montée du courant dans les enroulements. Nous ne parlerons pas ici de ces dispositifs.

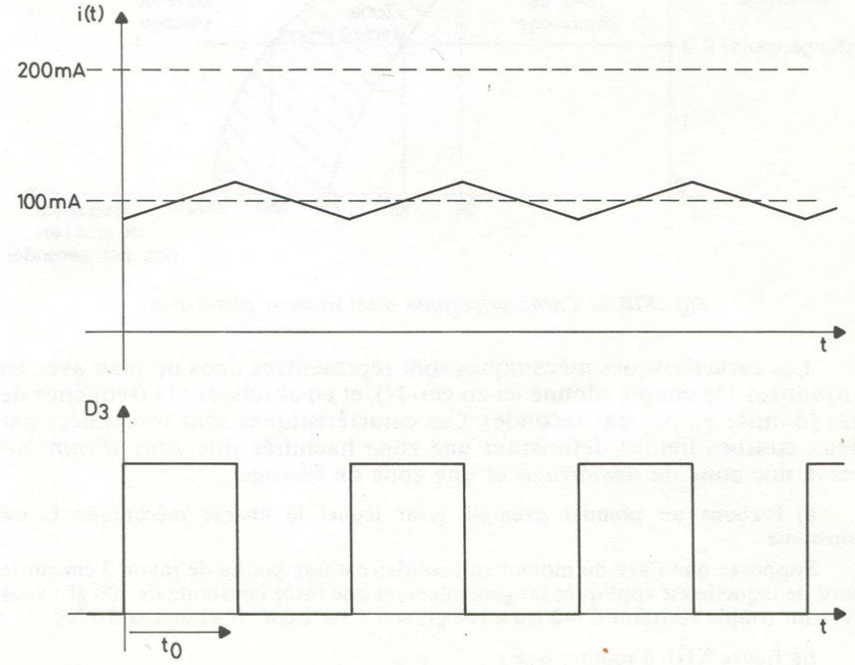


Fig. XIII. 5. Commande à fréquence élévée

II. 2. Caractéristiques mécaniques d'un moteur pas-à-pas

En pratique, la vitesse maximale de rotation d'un moteur pas à pas est plutôt déterminée par la *charge mécanique* qu'il doit entraîner. Cette charge s'exprime en terme de *couple* dont l'unité du système international est le m·N (mètre. newton).

Pour connaître la vitesse maximale à laquelle on peut faire tourner un moteur pas à pas, il faut disposer des caractéristiques du moteur (dont un exemple est donné figure XIII. 6), et connaître la valeur maximale du couple C que doit entraîner le moteur.

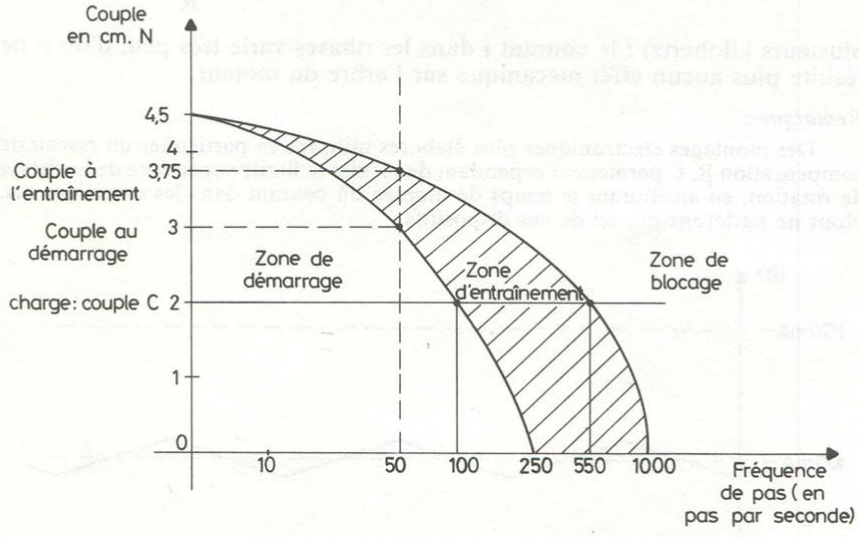


Fig. XIII. 6. Caractéristiques d'un moteur pas-à-pas

Les caractérisques mécaniques sont représentées dans un plan avec en ordonnées : le couple (donné ici en cm· N), et en abscisses : la fréquence de pas (donnée en pas par seconde). Ces caractéristiques sont constituées par deux courbes limites définissant une zone hachurée dite zone *d'entraînement*, une zone de *démarrage*, et une zone de *blocage*.

a) Prenons un premier exemple pour lequel la charge mécanique C est constante :

Supposez que l'axe du moteur soit solidaire d'une poulie de rayon 2 cm sur le bord de laquelle est appliquée tangentiellement une force constante de 100 gf : vous avez un couple résistant $C = 2\text{ cm} \times 100\text{ gf}$, soit $C \simeq 2\text{ cm} \cdot \text{N}$ (1 kg f $\simeq$ 10 N).

La figure XIII. 6 montre que :

— pour faire démarrer le moteur, on ne peut dépasser une fréquence de pas de

100 pas par seconde (soit environ 2 tours par seconde pour un moteur de 48 pas par tours) ; sinon, il y a perte de pas.

— ensuite, le moteur peut être accéléré jusqu'à 550 pas par seconde dans la zone d'entraînement ; si on dépasse cette fréquence, le moteur se bloque.

— si on désire arrêter le moteur sans perte de pas, il faut le décélérer jusqu'à 100 pas par seconde, *puis* arrêter les impulsions.

b) Prenons un deuxième exemple où cette fois on a décidé d'imposer une fréquence de pas constante de 50 pas par seconde :

La figure XIII. 6 montre que :

— au démarrage, on dispose sur l'arbre du moteur d'un couple (couple au démarrage) de 3 cm · N. Autrement dit, le moteur ne peut démarrer sans perte de pas que si la charge est au plus égale à 3 cm · N ;

— ensuite, le moteur tournant, la charge peut augmenter jusqu'à 3,75 cm · N ;

— si on désire arrêter le moteur sans perte de pas, il faut réduire la charge à une valeur inférieure ou égale à 3 cm · N, puis arrêter les impulsions.

II. 3. Commande par microprocesseur

a) *Conception générale de la commande*

D'un point de vue matériel, la façon la plus simple de commander un moteur pas à pas quatre phases avec un microprocesseur, est de relier les quatre points D_3, D_2, D_1, D_0 de la figure XIII. 3 à quatre sorties du registre CDP 1852 de la maquette VILÉMIO 4. Dans un premier temps nous établissons les connexions entre DO3 et D_3, DO2 et D_2, DO1 et D_1, DO0 et D_0. Les sorties DO3, DO2, DO1, et DO0 correspondent donc respectivement aux demi-enroulements P, Q, R et S.

Il apparaît que c'est l'exécution successive d'instructions OUTPUT 1, de code 61, qui provoquera la rotation du moteur :

— la figure XIII. 2 montre que la rotation dans le sens anti-horaire (A.H.) sera effectuée en transférant cycliquement vers (DO3, DO2, DO1, DO0) la succession des quatre quartets suivants :

{ 1,0,1,0) ; (0,1,1,0) ; (0,1,0,1) ; (1,0,0,1) } ou en notation hexadécimale :
{ A ; 6 ; 5 ; 9 }

— la rotation dans le sens horaire (H.) sera effectuée en transférant la succession suivante :

{ (1,0,0,1) ; (0,1,0,1) ; (0,1,1,0) ; (1,0,1,0) } ou, en notation hexadécimale :
{ 9 ; 5 ; 6 ; A }

b) *Exemple simple*

Soit à faire tourner le moteur dans le sens A.H. d'un nombre de pas égal à W_p. Nous supposons que l'utilisateur a introduit (avec l'aide du moniteur, ou par l'exécution d'un programme préalable, ou par tout autre moyen) le nombre W_p dans la mémoire à l'adresse 0C10 pointée par le registre R (0) par exemple (fig. XIII. 7).

100 pas par seconde (soit environ 2 tours par seconde pour un moteur de 48 pas par tours) ; sinon, il y a perte de pas.

— ensuite, le moteur peut être accéléré jusqu'à 550 pas par seconde dans la zone d'entraînement ; si on dépasse cette fréquence, le moteur se bloque.

— si on désire arrêter le moteur sans perte de pas, il faut le décélérer jusqu'à 100 pas par seconde, *puis* arrêter les impulsions.

b) Prenons un deuxième exemple où cette fois on a décidé d'imposer une fréquence de pas constante de 50 pas par seconde :

La figure XIII. 6 montre que :

— au démarrage, on dispose sur l'arbre du moteur d'un couple (couple au démarrage) de 3 cm · N. Autrement dit, le moteur ne peut démarrer sans perte de pas que si la charge est au plus égale à 3 cm · N ;

— ensuite, le moteur tournant, la charge peut augmenter jusqu'à 3,75 cm · N ;

— si on désire arrêter le moteur sans perte de pas, il faut réduire la charge à une valeur inférieure ou égale à 3 cm · N, puis arrêter les impulsions.

II. 3. Commande par microprocesseur

a) *Conception générale de la commande*

D'un point de vue matériel, la façon la plus simple de commander un moteur pas à pas quatre phases avec un microprocesseur, est de relier les quatre points D_3, D_2, D_1, D_0 de la figure XIII. 3 à quatre sorties du registre CDP 1852 de la maquette VILÉMIO 4. Dans un premier temps nous établissons les connexions entre DO3 et D_3, DO2 et D_2, DO1 et D_1, DO0 et D_0. Les sorties DO3, DO2, DO1, et DO0 correspondent donc respectivement aux demi-enroulements P, Q, R et S.

Il apparaît que c'est l'exécution successive d'instructions OUTPUT 1, de code 61, qui provoquera la rotation du moteur :

— la figure XIII. 2 montre que la rotation dans le sens anti-horaire (A.H.) sera effectuée en transférant cycliquement vers (DO3, DO2, DO1, DO0) la succession des quatre quartets suivants :

{ 1,0,1,0) ; (0,1,1,0) ; (0,1,0,1) ; (1,0,0,1) } ou en notation hexadécimale :
{ A ; 6 ; 5 ; 9 }

— la rotation dans le sens horaire (H.) sera effectuée en transférant la succession suivante :

{ (1,0,0,1) ; (0,1,0,1) ; (0,1,1,0) ; (1,0,1,0) } ou, en notation hexadécimale :
{ 9 ; 5 ; 6 ; A }

b) *Exemple simple*

Soit à faire tourner le moteur dans le sens A.H. d'un nombre de pas égal à W_p. Nous supposons que l'utilisateur a introduit (avec l'aide du moniteur, ou par l'exécution d'un programme préalable, ou par tout autre moyen) le nombre W_p dans la mémoire à l'adresse 0C10 pointée par le registre R (0) par exemple (fig. XIII. 7).

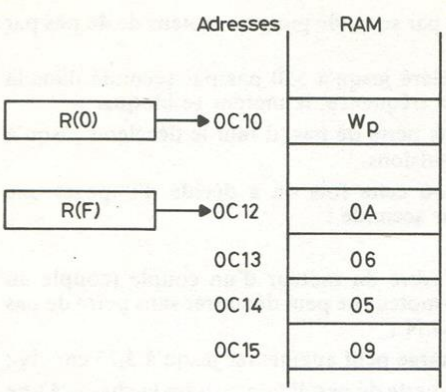


Fig. XIII. 7. Zone RAM pointée par R(0) et R(F)

Pour transférer vers (DO3, DO2, DO1, DO0) la suite des quartets (A, 6, 5, 9), ceux-ci peuvent être en mémoire à partir de l'adresse 0C12 (pourquoi pas) pointée par le registre R (F) (fig. XIII. 7).

La figure XIII. 8 donne alors l'organigramme de rotation du moteur dans le sens A.H. de W_p pas. Donnons quelques commentaires :

— comme nous savons que lors du premier pas nous transférons vers (DO3, DO2, DO1, DO0) le quartet (1,0,1,0) soit A en hexadécimal, nous initialisons ces 4 sorties avec la combinaison qui précède normalement celle-ci lors d'une rotation continue, c'est-à-dire avec le quartet ((1,0,0,1) soit 9 en hexadécimal. Ceci est réalisé en utilisant l'instruction OUT 1 dans le mode d'adressage immédiat, ce qui nécessite de rendre X égal à P (soit 3) par l'instruction SEX R (3) ;

— on remarque que le registre R (0) est aussi utilisé comme registre *compteur de pas ;*

— pour le bloc TEMPO, nous utilisons un sous-programme contenu dans l'EPROM moniteur, qui commence à l'adresse 0100. Ce sous-programme constitue une « pseudo-instruction » de format 5 octets ; rappelons que ceux-ci forment la suite D4. 01. 00. W1. W0, W1 et W0 définissant la durée $\theta = 24 \times (256.\ W1 + W0)$ microsecondes (la fréquence du quartz est de 2 MHz, et W1 et W0 sont en décimal dans cette formule).

Pour nos essais, nous avons choisi W1 = 01 et W0 = 00, ce qui donne θ de l'ordre de 6 ms, soit une fréquence de pas d'environ 150 Hz, soit 3 tours par seconde pour le moteur de 48 pas par tours que nous utilisons. Les caractéristiques de la figure 6 montrent que cette vitesse est compatible avec la charge de $1\ cm \cdot N$ qui était imposée à notre moteur.

Le tableau XIII. 1 donne le programme correspondant à l'organigramme de la figure XIII. 8. Pour le lancer, il faut afficher l'adresse 0400, puis frapper sur la touche P. Lorsque la rotation est terminée, le moniteur reprend la main et affiche « prêt ».

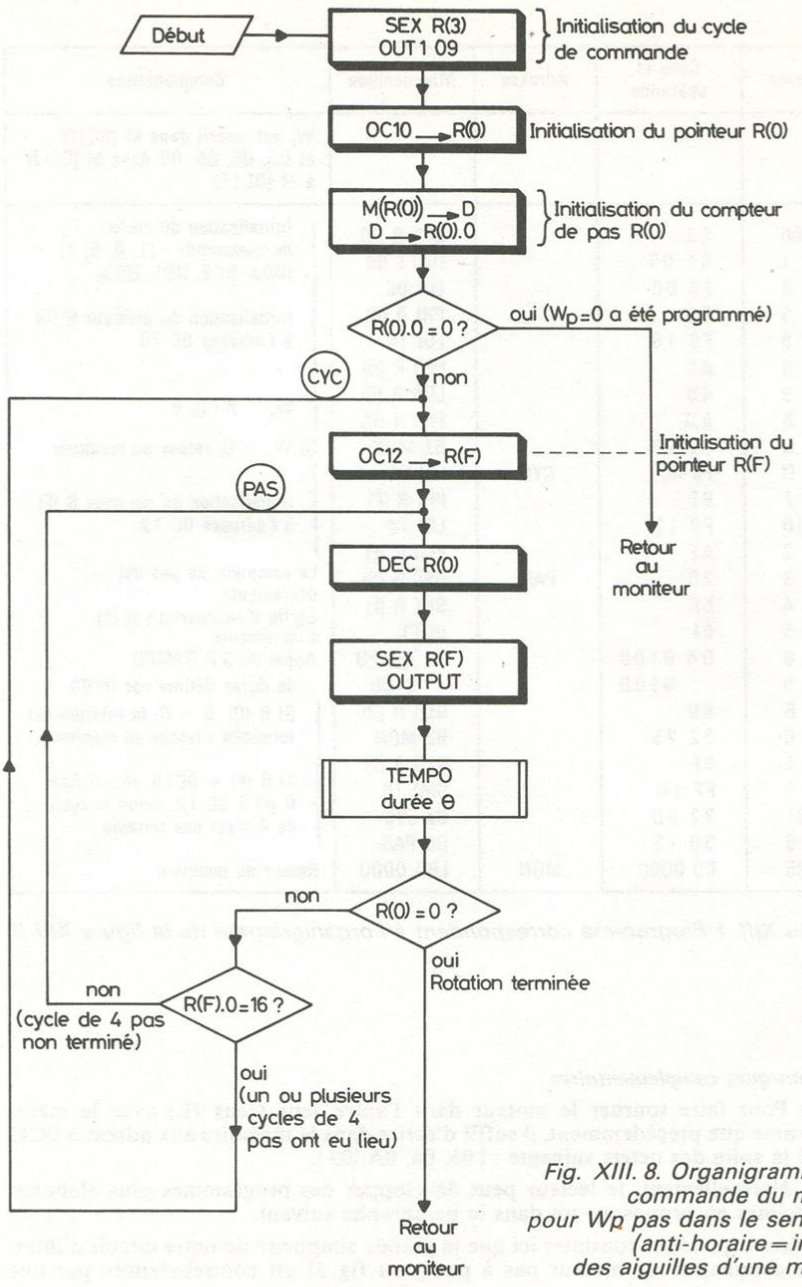


Fig. XIII. 8. Organigramme de commande du moteur pour Wp pas dans le sens A.H. (anti-horaire = inverse des aiguilles d'une montre)

Adresse	Code et opérande	Adresse	Mnémonique	Commentaires
				W_P est inscrit dans M (0C10) : et 0A, 06, 05, 09 dans M (0C12) à M (0C15).
0400	E3		SEX R (3)	Initialisation du cycle de commande : (1, 0, 0, 1) (D03, D02, D01, D00)
1	61 09		OUT1 09	
3	F8 0C		LDI 0C	Initialisation du pointeur R (0) à l'adresse 0C 10
5	B0		PHI R (0)	
6	F8 10		LDI 10	
8	A0		PLO R (0)	
9	40		LDA R (0)	W_p R (0). 0
A	A0		PLO R (0)	
B	32 25		BZ MON	Si W_p = 0, retour au moniteur
D	F8 0C	CYC	LDI 0C	Initialisation du pointeur R (F) à l'adresse 0C 12
F	BF		PHI R (F)	
10	F8 12		LDI 12	
2	AF		PLO R (F)	
3	20	PAS	DEC R (0)	Le compteur de pas est décrémenté
4	EF		SEX R (F)	
5	61		OUT1	Sortie d'un quartet ; R (F) s'incrémente
6	D4 0100		SP TEMPO	Appel du S.P. TEMPO
9	0100		0100	... de durée définie par 0100
B	80		GLO R (0)	Si R (0). 0 = 0, la rotation est terminée : retour au moniteur
C	32 25		BZ MON	
E	8F		GLO R (F)	Si R (F) = 0C16, réinitialiser R (F) à 0C 12, sinon le cycle de 4 n'est pas terminé
F	FF 16		SMI 16	
21	32 0D		BZ CYC	
3	30 13		BR PAS	
0425	C0 0000	MON	LBR 0000	Retour au moniteur

Tableau XIII. 1 Programme correspondant à l'organigramme de la figure XIII-8

c) *Remarques complémentaires*

α) Pour faire tourner le moteur dans l'autre sens (sens H.) avec le même programme que précédemment, il suffit d'écrire dans la mémoire aux adresses 0C12 à 0C16 la suite des octets suivante : { 05, 06, 0A, 09 }.

β) Naturellement, le lecteur peut développer des programmes plus élaborés. Nous-mêmes en proposons un dans le paragraphe suivant.

Il faut cependant constater ici que la grande simplicité de notre circuit d'interface de commande de moteur pas à pas (voir fig. 3) est contrebalancée par une

certaine complexité de la structure logicielle (du programme). En effet, ayez sous les yeux les figures 7 et 8, et supposez que vous désiriez faire tourner le moteur de W pas dans le sens A.H., puis de W' pas dans le sens H. Quelle est la procédure à suivre, si l'on désire utiliser la même structure de programme que celle proposée en figure 8 ?

— si W est divisible par 4 (on dit : égal à 0 modulo 4), le registre R (F) pointera la mémoire à l'adresse 0C12 lorsque les W pas dans le sens A.H. seront effectués. Ensuite, pour que le moteur tourne dans le sens H., une routine devra au préalable transférer dans la mémoire, à partir de l'adresse 0C12, la suite des 4 octets :

$Q_0 = (05,06,0A,09)$;

— mais si W n'est pas égal à 0 modulo 4, le registre R (F) pointera la mémoire, lorsque les W pas dans le sens A.H. seront effectués, à l'adresse :

0C13 si W = 1 modulo 4
0C14 si W = 2 modulo 4
0C15 si W = 3 modulo 4

Ainsi, pour que le moteur tourne dans le sens H. (après rotation de W pas dans le sens A.H.), une routine devra transférer dans la mémoire, à partir de l'adresse 0C12, la suite des 4 octets :

$Q_1 = (09,05,06,0A)$ si W = 1 modulo 4
$Q_2 = (0A,09,05,06)$ si W = 2 modulo 4
$Q_3 = (06,0A,09,05)$ si W = 3 modulo 4

La programmation de cette procédure demande une certaine attention, et complique la vie du programmeur si le microprocesseur a par ailleurs d'autres tâches à accomplir.

On peut se débarrasser de cette contrainte intellectuelle en acceptant de compliquer légèrement la structure matérielle de notre circuit de commande de moteur pas à pas. C'est l'objet du paragraphe suivant.

III. Commande d'un moteur pas à pas. Variante II

III. 1. Un circuit de commande commode

Le circuit que nous proposons maintenant est présenté figure XIII. 9. Il utilise un compteur CD 4029 (RCA) et une quadruple porte « OU EXCLUSIF » CD 4030 (RCA).

Le compteur CD 4029 fonctionne de la façon suivante :

— si PE = 0 et CI = 0, un front montant sur l'entrée Cl. incrémente le mot (A_3, A_2, A_1, A_0) si $C/\overline{D} = 1$, et décrémente ce mot si $C/\overline{D} = 0$;

— si $BIN/\overline{DEC} = 1$, le comptage ou décomptage s'effectue en binaire, sinon en décimal ;

— si PE = 1, le compteur est transparent, c'est-à-dire que (A_3, A_2, A_1, A_0) = (J_3, J_2, J_1, J_0). Naturellement, des impulsions sur Cl. sont alors sans effet.

Comme le montre la figure XIII.9, PE et CI sont imposés au niveau logique 0, $BIN/\overline{DEC}$ au niveau logique 1. A_3 et A_2 ne sont pas connectés.

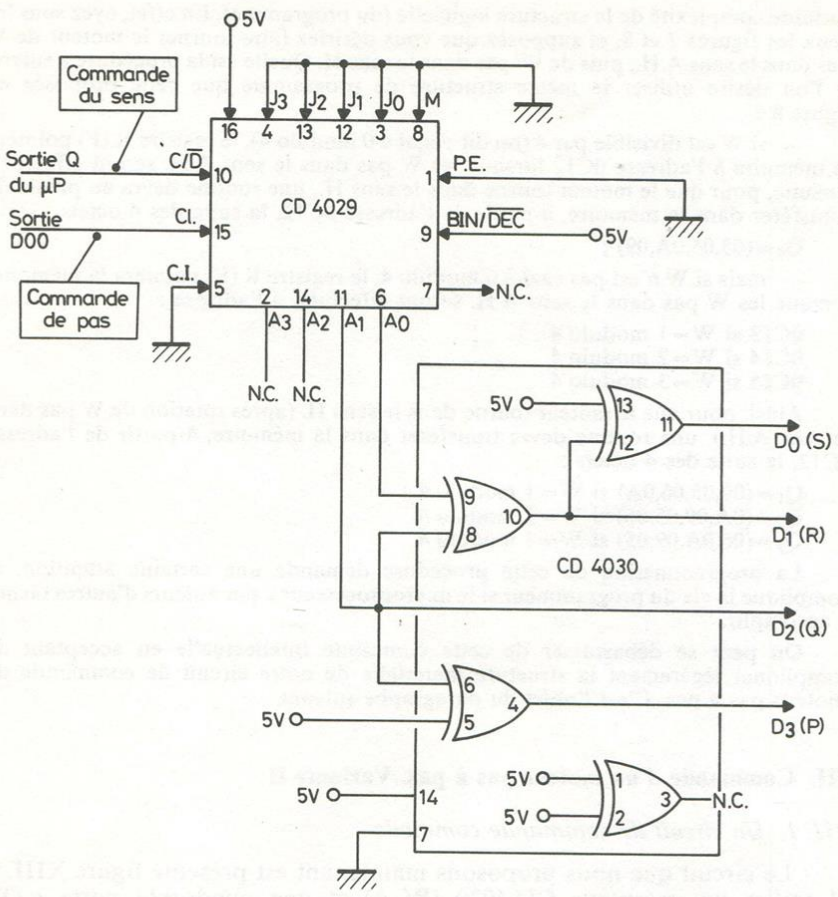


Fig. XIII. 9. Circuit de commande proposé

Ainsi, lorsque $C/\overline{D} = 1$ et que des fronts sont appliqués sur l'entrée Cl., il apparaît sur (A_1, A_0) la séquence suivante :

A_1, A_0 = ---- ; (0,0) ; (0,1) ; (1,0) ; (1,1) ; (0,0) ;---

Mais si $C/\overline{D} = 0$, la séquence est la suivante :

(A_1, A_0) = ---- ; (0,0) ; (1,1) ; (1,0) ; (0,1) ; (0,0) ;---

Par ailleurs, les liaisons avec les « OU EXCLUSIF » imposent pour D_3, D_2, D_1, D_0 les équations logiques suivantes :

$$D_3 = \overline{A_1}$$
$$D_2 = A_1$$
$$D_1 = A_1 \oplus A_0$$
$$D_0 = \overline{A_1 \oplus A_0}$$

Ainsi, lorsque $C/\overline{D} = 1$ et que des fronts sont appliqués sur l'entrée Cl. ; il apparaît sur (D_3, D_2, D_1, D_0) la séquence suivante :

$$(D_3,D_2,D_1,D_0) = \ldots ; \underbrace{(1,0,0,1)}_{9} ; \underbrace{(1,0,1,0)}_{A} ; \underbrace{(0,1,1,0)}_{6} ; \underbrace{(0,1,0,1)}_{5} ; \ldots$$

Mais si $C/D = 0$, la séquence est la suivante :

$$(D_3,D_2,D_1,D_0) = \ldots ; \underbrace{(1,0,0,1)}_{9} ; \underbrace{(0,1,0,1)}_{5} ; \underbrace{(0,1,1,0)}_{6} ; \underbrace{(1,0,1,0)}_{A} ; \ldots$$

Il apparaît que l'entrée $C/\overline{D}$ est la commande de sens de rotation du moteur : pour $C/\overline{D} = 1$, le moteur tourne dans le sens A.H., sinon dans le sens H. L'entrée Cl. est la commande de rotation d'un pas.

On comprend qu'avec ce montage, la routine spécifique destinée à commander la rotation du moteur, risque d'être simple. Pour l'exemple que nous proposons, nous avons *relié $C/\overline{D}$ à la sortie Q du* microprocesseur, et *Cl. à la sortie DO0* de la maquette entrée/sortie. Remarquons que nous n'avons pas besoin de maintenir un *niveau* sur l'entrée Cl., et que par conséquent on pourrait directement relier cette entrée à la sortie N_0 du microprocesseur.

III. 2. Position d'un problème et sa résolution :

a) *But à atteindre :*

Nous désirons être capable de commander le moteur directement à partir du clavier de la maquette « clavier-afficheur ». En outre, nous désirons utiliser les quatre afficheurs (ceux qui indiquent « prêt » après initialisation) de cette même maquette, pour programmer le sens de rotation et le nombre de pas que devra effectuer le moteur.

Plus précisément, la figure XIII. 10 indique la signification que nous décidons de donner aux quatre caractères affichés :

— le caractère $[_1$ indique le sens de rotation :
si $[_1 = 0$, le sens doit être horaire
si $[_1 = 8$, le sens doit être anti-horaire
autrement dit, le programme doit mettre la sortie Q du microprocesseur à 0 si $[_1 = 0$, et à 1 si $[_1 = 8$.

— Le caractère $[_2$ indique le nombre de *tours* que doit effectuer le moteur.

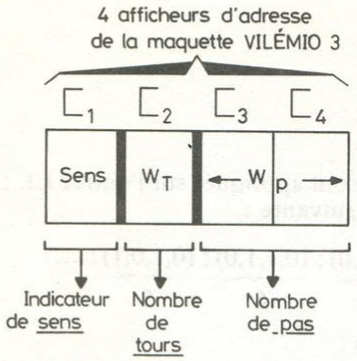


Fig. XIII. 10. Signification des afficheurs

Autrement dit, le programme doit envoyer sur DO0, c'est-à-dire sur l'entrée Cl. de notre circuit d'interface, $48 \times W_T$ impulsions.

— L'ensemble des caractères ([3 [4) indique le nombre de *pas,* en *décimal,* que doit effectuer le moteur, *en plus* du nombre de tours indiqué par le caractère [2.

Ainsi par exemple, si l'utilisateur tape ([1, [2, [3, [4 = (0,9,7,4), notre but est de faire comprendre au microprocesseur qu'il s'agit de faire tourner le moteur dans le sens horaire de 9 tours et de 74 pas.

Enfin, l'ordre d'exécution consistera en la pression de la touche utilisateur U.

b) *Aide apportée par le moniteur*

— si vous avez pris le soin de lire le chapitre concernant l'étude du programme moniteur, vous avez dû remarquer que *le registre R (A)* de 16 bits, interne au microprocesseur, contient précisément (en binaire naturellement) les quatre caractères hexadécimaux [1, [2, [3, [4 inscrits sur les quatre afficheurs d'adresse. Ainsi, pour que le microprocesseur prenne en compte ces quatre caractères, il lui suffit d'examiner le contenu du registre R (A) ;

— pour compter le nombre de pas W_p dont le moteur doit tourner, nous utilisons un registre (le registre R (7)) que l'on décrémente à chaque pas effectué. Le contenu initial de ce registre doit être la valeur W_p en *hexadécimal.*

Or nous entrons W_p en *décimal* sur les caractères [3 et [4. Une conversion décimal → hexa est donc nécessaire ; celle-ci peut être avantageusement réalisée par l'intermédiaire du sous-programme écrit à l'adresse 03C5 de l'EPROM moniteur.

Rappelons que ce sous-programme constitue une « pseudo-instruction » de format 5 octets : ces 5 octets forment la suite D4.03.C5.nl.n0, n1.n0 étant *l'adresse* du mot de 8 bits que l'on désire convertir en hexadécimal ; le résultat est transféré dans la mémoire à cette même adresse.

Nous avons choisi n1.n0 = 0C.10.

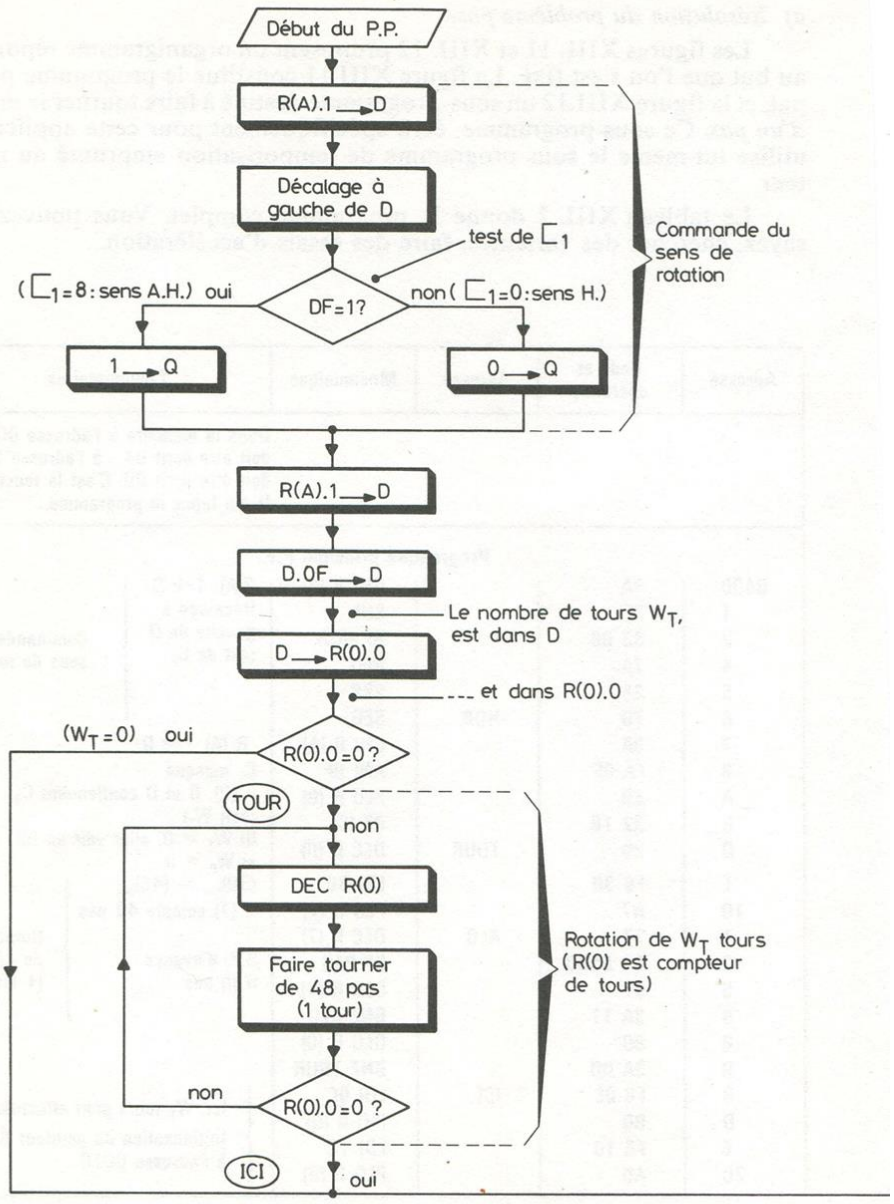
c) *Résolution du problème posé*

Les figures XIII. 11 et XIII. 12 proposent un organigramme répondant au but que l'on s'est fixé. La figure XIII.11 constitue le programme principal, et la figure XIII.12 un sous-programme destiné à faire tourner le moteur *d'un pas.* Ce sous-programme, écrit spécifiquement pour cette application, utilise lui-même le sous-programme de temporisation emprunté au moniteur.

Le tableau XIII. 2 donne le programme complet. Vous pouvez l'essayez, chercher des variantes, faire des essais d'accélération...

Adresse	Code et opérande	Adresse	Mnémonique	Commentaires
				Dans la mémoire à l'adresse 0C74 doit être écrit 04 : à l'adresse 0C75 doit être écrit 00. C'est la touche U qui lance le programme.
			Programme principal P.P.	
0400	9A		GHI R (A)	R(A). 1 → D
1	FE		SHL	Décalage à gauche de D
2	33 06		BF HOR	test de C_1 — Commande du sens de rotation
4	7A		REQ	
5	38		SKP	
6	7B	HOR	SEQ	
7	9A		GHI R (A)	R (A) 1 → D
8	FA 0F		ANI 0F	C_1 masqué
A	A0		PLO R (0)	R (0). 0 et D contiennent C_2 (soit W_T)
B	32 1B		BZ ICI	Si W_T = 0, aller voir en ICI
D	20	TOUR	DEC R (0)	si W_P = 0
E	F8 30		LDI 30	$(30)_{hex} = (48)_{dec}$
10	A7		PLO R (7)	R (7) compte 48 pas
1	27	ALO	DEC R (7)	— Rotation de 48 pas (1 tour)
2	D4 0500		SP PAS	S.P. d'avance
5	87		GLO R (7)	d'un pas
6	3A 11		BNZ ALO	
8	80		GLO R (0)	
9	3A 0D		BNZ TOUR	
B	F8 0C	ICI	LDI 0C	Ici, W_T tours sont effectués
D	B0		PHI R (D)	
E	F8 10		LDI 10	Initialisation du pointeur R (0)
20	A0		PLO R (D)	à l'adresse 0C10

(Suite page 280)


Début du P.P.
R(A).1 → D
Décalage à gauche de D
test de ⊏1
Commande du sens de rotation
(⊏1 = 8 : sens A.H.) oui
DF=1?
non (⊏1 = 0 : sens H.)
1 → Q
0 → Q
R(A).1 → D
D.0F → D
Le nombre de tours WT, est dans D
D → R(0).0
et dans R(0).0
(WT = 0) oui
R(0).0 = 0 ?
TOUR
non
DEC R(0)
Faire tourner de 48 pas (1 tour)
Rotation de WT tours (R(0) est compteur de tours)
non
R(0).0 = 0 ?
ICI
oui

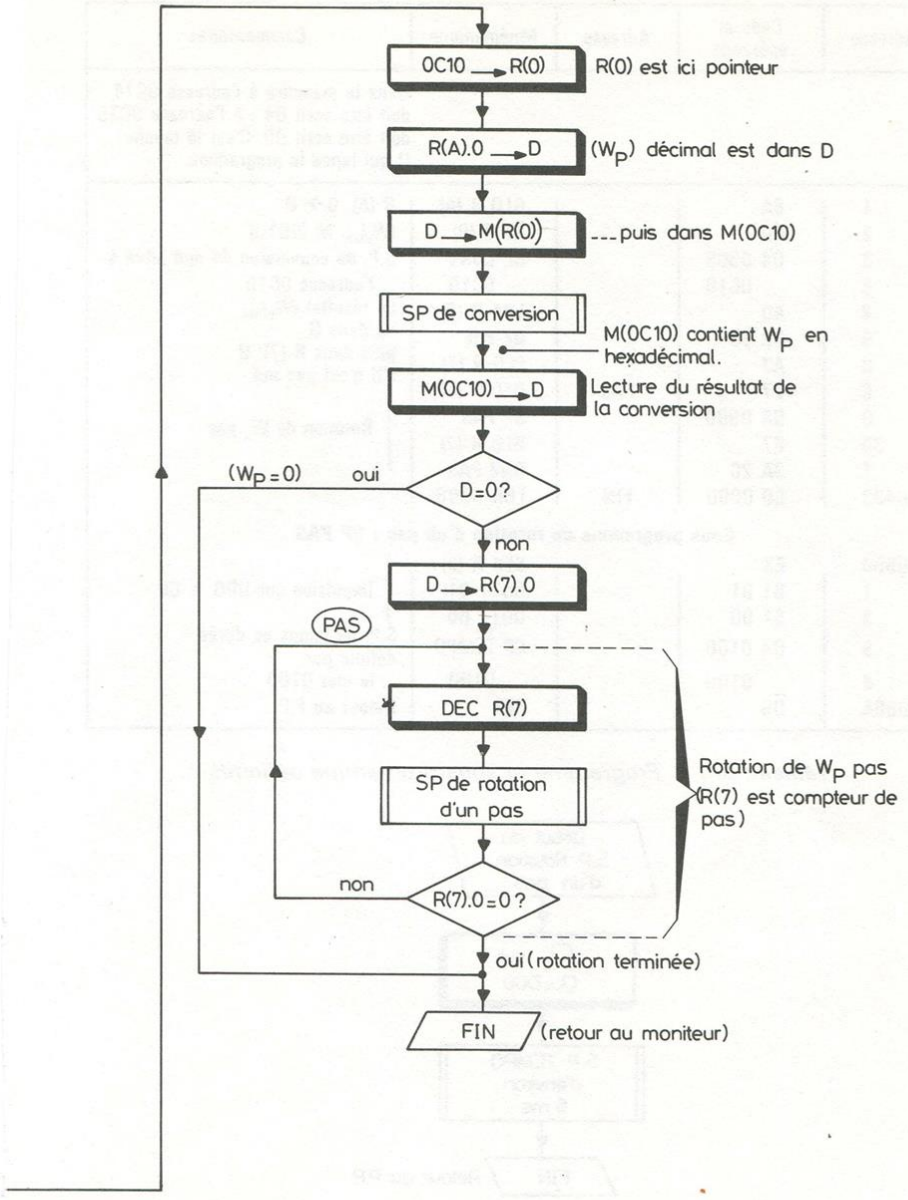


Fig. XIII. 11. Organigramme correspondant

Adresse	Code et opérande	Adresse	Mnémonique	Commentaires
				Dans la mémoire à l'adresse 0C74 doit être écrit 04 : à l'adresse 0C75 doit être écrit 00. C'est la touche U qui lance le programme.
1	8A		GLO R (A)	R (A). 0 → D
2	50		ST R (0)	$(W_p)_{dec}$ M (0C10)
3	D4 03C5		SP CONV	S.P. de conversion du mot situé à
6	0C10		0C10	... l'adresse 0C10
8	40		LDA R (0)	Le résultat $(W_p)_{hex}$
9	32 33		BZ FIN	va dans D
B	A7		PLO R (7)	puis dans R (7). 0
C	27	PAS	DEC R (7)	s'il n'est pas nul
D	D4 0500		SP PAS	} Rotation de W_p pas
30	87		GLO R (7)	
1	3A 2C		BNZ PAS	
0433	C0 0000	FIN	LBR 0000	
	Sous programme de rotation d'un pas : SP PAS			
0500	E3		SEX R (3)	} Impulsion sur D00 = CL
1	61 01		OUT1 01	
3	61 00		OUT1 00	
5	D4 0100		SP TEMPO	S.P. de temps de durée définie par
8	0100		0100	... le mot 0100
050A	D5		RET	Retour au P.P.

Tableau XIII. 2. Programme et sous-programme définitifs

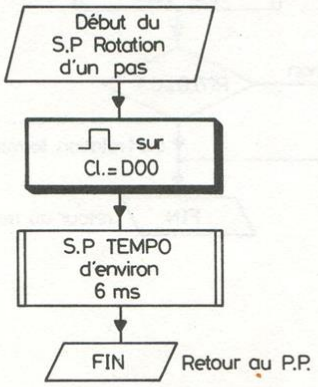


Fig. XIII. 12. Organigramme du S.P. de rotation d'un pas

Adresse	Code et opérande	Adresse	Mnémonique	Commentaires
				Dans la mémoire à l'adresse 0C74 doit être écrit 04 : à l'adresse 0C75 doit être écrit 00. C'est la touche U qui lance le programme.
1	8A		GLO R (A)	R (A). 0 → D
2	50		ST R (0)	$(W_p)_{dec}$ M (0C10)
3	D4 03C5		SP CONV	S.P. de conversion du mot situé à
6	0C10		0C10	... l'adresse 0C10
8	40		LDA R (0)	Le résultat $(W_p)_{hex}$
9	32 33		BZ FIN	va dans D
B	A7		PLO R (7)	puis dans R (7). 0
C	27	PAS	DEC R (7)	s'il n'est pas nul
D	D4 0500		SP PAS	} Rotation de W_p pas
30	87		GLO R (7)	
1	3A 2C		BNZ PAS	
0433	C0 0000	FIN	LBR 0000	
	Sous programme de rotation d'un pas : SP PAS			
0500	E3		SEX R (3)	} Impulsion sur D00 = CL
1	61 01		OUT1 01	
3	61 00		OUT1 00	
5	D4 0100		SP TEMPO	S.P. de temps de durée définie par
8	0100		0100	... le mot 0100
050A	D5		RET	Retour au P.P.

Tableau XIII. 2. Programme et sous-programme définitifs

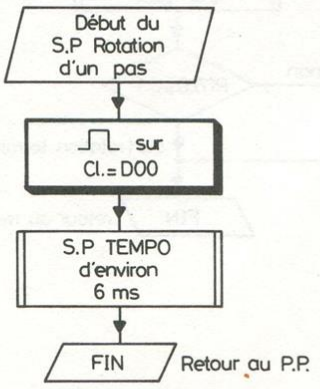


Fig. XIII. 12. Organigramme du S.P. de rotation d'un pas

d) *Mise en œuvre de l'ensemble*

1. Mettre en route le système VILÉMIO et relier la sortie Q du microprocesseur à l'entrée C/$\overline{D}$ du compteur CD 4029, la sortie DO0 de la maquette entrée/sortie à l'entrée Cl. du compteur.

2. Ne pas oublier d'appliquer la tension E = 12 V au point commun des enroulements du moteur. (A défaut, on peut prendre E = 5 V, mais le couple disponible sera plus faible).

3. Ecrire le P.P. à partir de l'adresse 0400, et le S.P. PAS à partir de l'adresse 0500.

4. Ecrire dans la mémoire aux adresses 0C74 et 0C75 l'adresse du début du programme, soit respectivement les octets 04 et 00.

5. Initialiser le système en frappant la touche I : « prêt » s'affiche.

6. Programmer sur les afficheurs d'adresse le sens de rotation et le nombre de tours et de pas souhaités.

7. Frapper la touche U : le programme s'exécute (le moteur tourne dans le sens souhaité et effectuant le nombre de tours et de pas programmés).

8. Lorsque le moniteur retrouve la main, reprendre à partir de l'opération n° 6.

Chapitre 14
LA MESURE DES GRANDEURS PHYSIQUES

I. Introduction

Pour qu'un microprocesseur puisse « traiter » une grandeur physique G, il faut bien sûr que celle-ci soit, d'une façon ou d'une autre, introduite dans le microprocesseur sous la forme d'un nombre binaire W.

Une grandeur physique est une grandeur de caractère *continu* (par opposition au caractère *discret* d'un nombre binaire). Ce chapitre a essentiellement pour but de présenter et d'étudier divers moyens plus ou moins avantageux, pour transformer une grandeur continue (on dit encore « analogique ») en une grandeur discrète (on dit encore « numérique ») : ceci s'appelle la conversion analogique-numérique ou CAN.

Cependant, pour transformer la grandeur G en son image numérique W, il faut élaborer au préalable l'image *électrique* U (tension) de cette grandeur ; ce rôle incombe à un transducteur (capteur) dont le fonctionnement utilise un principe de physique concernant la grandeur G. Par exemple, si G est une température T, le principe du capteur utilise le fait que la tension de seuil d'une diode varie avec T ; dans ce cas la fonction $U = f(G)$ (fig. XIV. 1) est linéaire dans une certaine gamme de températures. Si G est un champ magnétique B, le capteur utilise l'effet Hall ; le capteur s'appelle alors sonde à effet Hall, et fournit une tension U encore linéaire avec B.

L'élaboration de W, image numérique de G, constitue la *mesure* de G si la relation entre W et G est connue ; cette relation est souvent matérialisée par une expression mathématique linéaire, sinon par une courbe obtenue expérimentalement.

Compte tenu du caractère discret de W, la mesure de G conduit nécessairement à une erreur systématique. En effet, si W_1 et W_2 sont les images numériques respectivement de G_1 et G_2, on ne peut distinguer G_1 de G_2 si $W_1 - W_2 < 1$ bit : on dit que la *résolution* est de 1 bit. Si le format de W est de 8 bits (ce sera le cas dans ce chapitre), la *précision* maximale de la mesure est de $1 : 2^8$, soit environ 0,4 %.

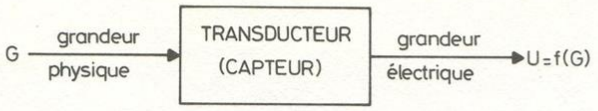


Fig. XIV. 1. Fonction d'un transducteur

II. La conversion numérique-analogique

II. 1. Fonction d'un convertisseur numérique-analogique (CNA)

Comme nous le verrons plus loin, la transformation d'une grandeur continue en une grandeur discrète utilise en fait la transformation réciproque, c'est-à-dire la CNA.

La fonction d'un CNA (voir fig. XIV. 2) est de transformer un mot binaire $W=(DO_7,\ldots,DO_0)$ en une grandeur analogique S (tension électrique), selon la relation suivante :

$$S = V_{REF}\left(\frac{DO_0}{2^8}+\frac{DO_1}{2^7}+\ldots+\frac{DO_6}{2^2}+\frac{DO_7}{2}\right) = V_{REF}\,\frac{(W)_{dec.}}{256}$$

Avec une tension de référence V_{REF} égale à + 10 V, nous trouvons que :

— si $W = (0,0,0,0,0,0,0,0)$ alors $S = 0$ V.

— si $W = (0,0,0,0,0,0,0,1)$ alors $S = \frac{V_{REF}}{256} = 0{,}039$ V.

Nous remarquons que la résolution de 1 bit équivaut à une résolution sur S de 39 mV :

— si $W = (1,0,0,0,0,0,0,0)$ alors $S = V_{REF}\frac{128}{256} = 5{,}000$ V.

— si $W = (1,1,1,1,1,1,1,1)$ alors $S = V_{REF}\frac{255}{256} = 9{,}961$ V.

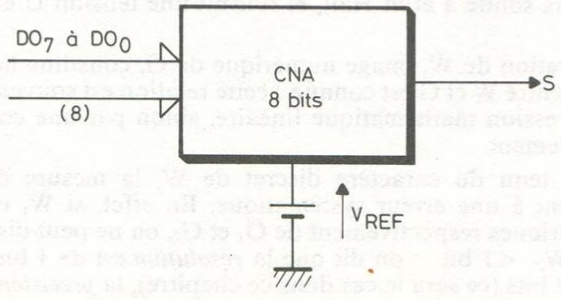


Fig. XIV. 2. Fonction d'un CNA

II. 2. Liaisons entre CNA et microprocesseur

Pour que le microprocesseur puisse générer une tension S, image d'un mot binaire W, vers le monde extérieur, il faut donc transférer ce mot W en entrée d'un CNA. Ceci doit être réalisé par l'intermédiaire d'un registre de sortie déclenché lors de l'exécution d'une instruction OUTPUT.

Supposez que les 8 sorties DO_7 à DO_0 de la maquette entrée/sortie soient reliées aux 8 entrées d'un CNA ; l'exécution de l'instruction OUT 1 (de code 61) aura pour effet de transférer en sortie du CNA une tension S, image du mot W situé en mémoire à l'adresse définie par le contenu du registre R (X).

Pour notre part, nous avons utilisé un boîtier DAC [1]-UP8BC de chez DATEL (fig. XIV. 3) contenant un CNA 8 bits, un registre 8 bits, et une tension de référence V_{REF}.

Le registre interne à ce boîtier est commandé par l'entrée $\overline{LOAD}$:

— si $\overline{LOAD} = 0$, le registre est transparent ;

— si $\overline{LOAD}$ passe au niveau logique 1, le registre mémorise la donnée qui était présente à son entrée lorsque $\overline{LOAD}$ était au niveau logique 0.

Il s'agit donc d'un registre « latch » dont l'existence n'est en fait pas indispensable si l'on décide d'utiliser la maquette entrée/sortie, puisque celle-ci contient déjà un registre de sortie ; dans ce cas, il faut imposer l'entrée $\overline{LOAD}$ au niveau logique 0.

Si l'on n'utilise pas la maquette entrée/sortie, les 8 entrées du registre interne au boîtier DAC-UP8BC sont à relier directement au bus des données ; dans ce cas, l'entrée $\overline{LOAD}$ doit être commandée par la sortie TPB du microprocesseur et l'une de ses lignes N (la ligne N_0 par exemple, comme le montre la figure XIV. 3). Ainsi, pendant l'exécution d'une instruction OUTPUT qui active la ligne N_0, $\overline{LOAD}$ passe au niveau 0 lorsque TPB passe au niveau 1 (c'est-à-dire lorsqu'on est sûr que le mot M (R(X)) est présent sur le bus des données), puis $\overline{LOAD}$ revient au niveau 1 lorsque TPB revient au niveau 0.

Le boîtier DAC nécessite une alimentation double de ± 15 V. La tension de sortie S peut varier de 0 à $+10$ V, ou de -5 V à $+5$ V ; dans le premier cas, on est dans le mode unipolaire, et dans le deuxième cas dans le mode bipolaire. Le choix du mode est effectué par l'intermédiaire de la broche n° 15 : si elle est « en l'air » (ce que nous avons choisi), le mode est unipolaire ; si elle est reliée à la broche n° 20, le mode est bipolaire.

La tension de référence intégrée peut-être ajustée par l'intermédiaire de la résistance variable notée *règlage du gain*. Pour régler cette tension de référence à 10 V, il faut, le mot W = (1,1,1,1,1,1,1,1) étant présenté en entrée du CNA, régler la tension de sortie S à la valeur 9,961 V.

La tension de sortie S est fournie par un amplicateur opérationnel intégré au boîtier, dont on peut ajuster le zéro par l'intermédiaire de la résistance variable notée

(1) DAC (digital to analog-converter) = CNA

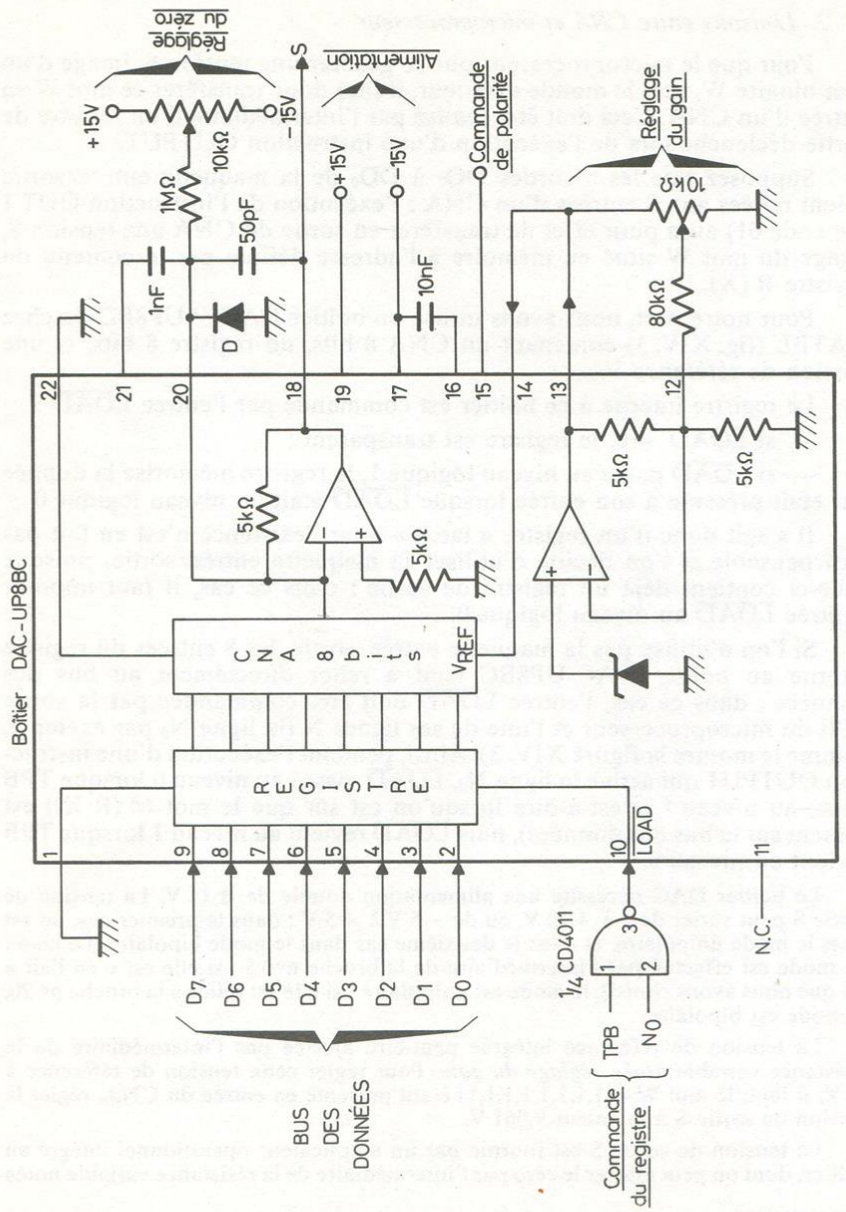


Fig. XIV. 3. Organisation du DAC-UP 8BC

réglage du zéro. Pour faire ce réglage, il faut, le mot W=(0,0,0,0,0,0,0,0,) étant présenté en entrée du CNA, régler la tension de sortie à la valeur 0 V.

Les condensateurs représentés sur la figure XIV. 3 servent à éviter l'entrée en oscillations de la sortie S. La diode D, qui peut être d'un type quelconque, protège l'entrée « moins » de l'amplificateur opérationnel dans le cas d'une fausse manœuvre.

III. La conversion analogique-numérique (CAN)

III. 1. Montage proposé

La figure XIV. 4 présente un montage qui permet, comme nous allons l'expliquer, de transformer une tension électrique U en son image numérique W, image finalement contenue dans la mémoire du système VILÉMIO. Ce montage utilise un comparateur (CA339 de chez RCA ou équivalent) dont la sortie est reliée à l'entrée $\overline{EF}_1$ du microprocesseur, et dont les entrées + et − sont respectivement la tension U à mesurer et la tension de sortie S du CNA commandé par le microprocesseur lui-même : si $S < U$, $\overline{EF}_1 = 1$ et si $S > U$, $\overline{EF}_1 = 0$. Notons par ailleurs que le comparateur présente un certain seuil de sensibilité de l'ordre du mV. Cela veut dire que si les tensions S et U sont tellement voisines que leur différence est inférieure à ce seuil, le comparateur ne réagit pas. Il est important de constater que l'existence de ce seuil n'introduit pas d'erreur supplémentaire dans la mesure de U puisque la sortie S du CNA présente une résolution de 39 mV, valeur supérieure au seuil du comparateur.

Appelons W (U) l'image numérique de U. Pour une tension de référence V_{REF} réglée à 10 V, W (U) est un mot binaire compris entre 00 et FF (en notation hexadécimale) si la tension U à mesurer est comprise entre 0 et 9,961 V ; ce que nous supposerons. Le microprocesseur se trouve en quelque sorte dans la position de « deviner » W (U). Pour ce faire, il doit « essayer » successivement diverses valeurs en envoyant sur l'entrée du CNA une séquence de mots W_1, W_2, W_3,... Lorsque le mot W_i est envoyé, le comparateur réagit en imposant un certain niveau logique sur sa sortie $\overline{EF}_1$: si W_i est inférieur à l'image numérique de U à trouver, cela veut dire que la tension de sortie S du CNA est inférieure à U, et par conséquent $\overline{EF}_1 = 1$; si W_i est supérieur à l'image numérique de U, alors $\overline{EF}_1 = 0$. En fonction de la réponse fournie par le comparateur, le microprocesseur peut envoyer en entrée du CNA un mot suivant W_{i+1}, plus proche de W (U) que le mot précédent W_i. Ainsi, au bout d'un certain temps qu'on cherche à rendre le plus petit possible, l'image numérique W (U) est atteinte.

La recherche de W (U) par le microprocesseur se fait selon deux méthodes : la méthode dite « linéaire », et la méthode dite « par approximations successives ».

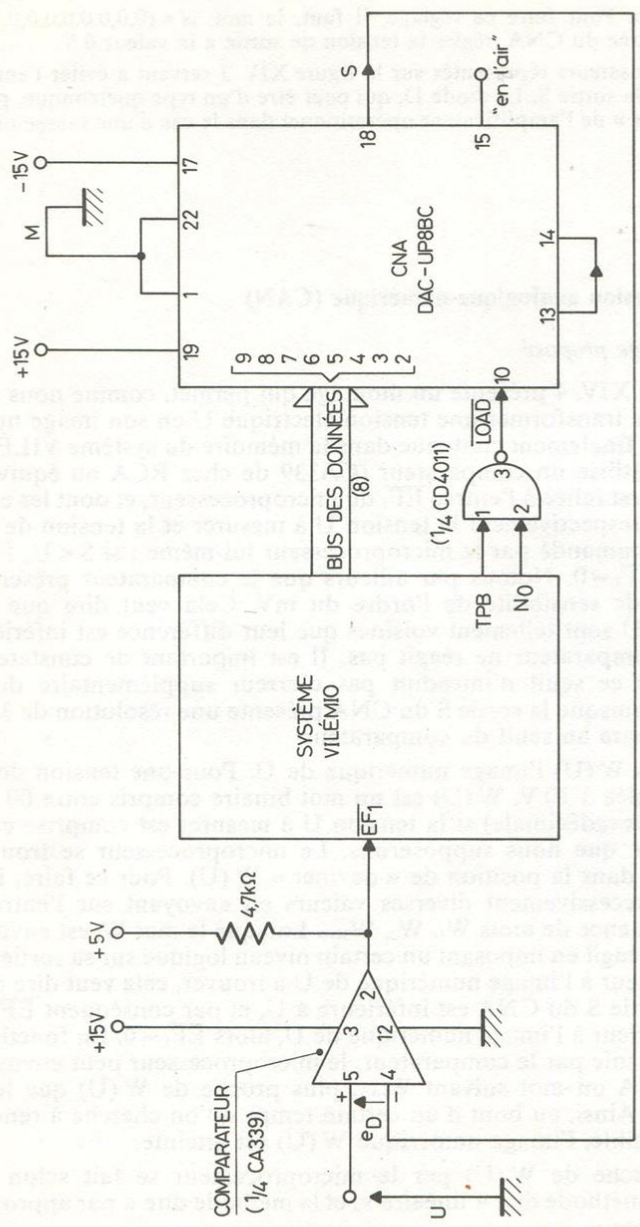


Fig. XIV. 4. Transformation d'une tension en image numérique W

III. 2. La CAN « linéaire »

Si vous voulez rechercher dans le dictionnaire un mot commençant par la lettre P par exemple, vous pouvez ouvrir le dictionnaire à la lettre A et feuilleter les pages une par une : vous finirez par trouver le mot cherché. Cette méthode de recherche, qui ne nécessite pas un gros effort intellectuel, a le seul mérite de fonctionner. Elle est simple à mettre en œuvre, et peut facilement être transposée à la recherche de l'image numérique de U par le microprocesseur.

Observez l'organigramme de la figure XIV. 5. Une ligne de mémoire RAM pointée par le registre R (X) (fig. XIV. 6) est choisie pour contenir un mot W à sortir vers le CNA. Au début, W = 00 ; ainsi, la première exécution de l'instruction OUTPUT a pour effet de fournir à l'entrée (−) du comparateur la tension S = 0V. Si U n'est pas nul, $\overline{EF_1} = 1$, et W = 01 ; ainsi, la

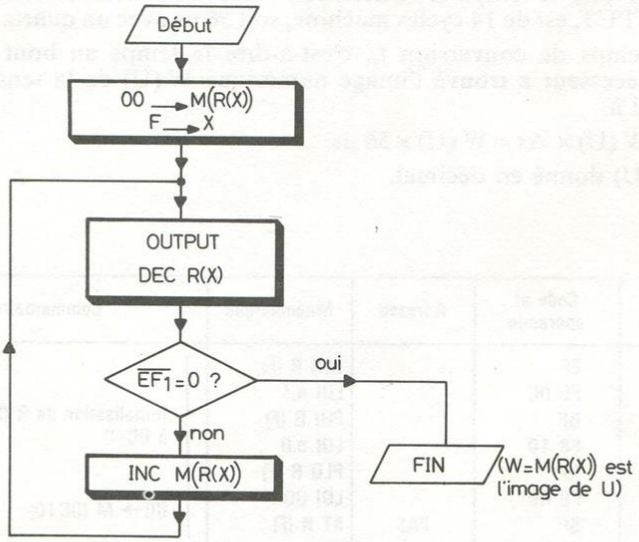


Fig. XIV. 6. Une ligne RAM est pointée par R(X)

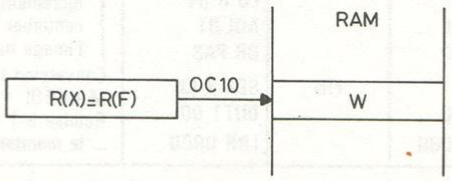


Fig. XIV. 5. Organigramme de « CAN linéaire »

deuxième exécution de l'instruction OUTPUT a pour effet de fournir la tension S = 39 mV. Si U est supérieur à 39 mV, $\overline{EF}_1 = 1$, et W = 02 ; ainsi, la troisième exécution de l'instruction OUTPUT a pour effet de fournir la tension S = 2 × 39 mV. Si U est supérieur à 2 × 39 mV, $\overline{EF}_1 = 1$, et W = 03. Le processus continue jusqu'à temps que la tension S dépasse la tension U à mesurer : la conversion est terminée.

La figure XIV.7 montre la forme de la tension S pendant le déroulement du programme ; il s'agit d'une fonction en escalier. La hauteur d'une marche est égale à la résolution $\frac{V_{REF}}{256} = 39$ mV. On notera la forme globalement linéaire de la courbe S (t) d'où le nom donné à ce procédé.

Le tableau XIV. 1 donne le programme permettant la mise en œuvre de cette méthode de conversion analogique-numérique. Son examen montre que l'intervalle de temps Δt entre deux exécutions successives de l'instruction OUTPUT, est de 14 cycles machine, soit 56 µs avec un quartz de 2 MHz.

Le temps de conversion t_c, c'est-à-dire le temps au bout duquel le microproccesseur a trouvé l'image numérique W (U) de la tension U, est donc égal à :

$t_c = W(U) \times \Delta t = W(U) \times 56\ \mu s$

avec W (U) donné en décimal.

Adresse	Code et opérande	Adresse	Mnémonique	Commentaires
0400	EF		SEX R (F)	
1	F8 0C		LDI n.1	
3	BF		PHI R (F)	Initialisation de R (X) = R (F) à 0C10
4	F8 10		LDI n.0	
6	AF		PLO R (F)	
7	F8 00		LDI 00	
9	5F	PAS	ST R (F)	00 → M (0C10)
A	61		OUT1	
B	2F		DEC R (F)	
C	34 13		B1 FIN	Si S ⩾ U, conversion terminée, sinon...
E	0F		LD R (F)	incrémenter M (0C10) et continuer la recherche de l'image numérique de U
F	FC 01		ADI 01	
11	30 09		BR PAS	
3	E3	FIN	SEX R (3)	Conversion terminée : M (0C10) = W (U)
4	61 00		OUT1 00	Remise à 1 de $\overline{EF}_1$ avant que...
16	C0 0000		LBR 0000	... le moniteur reprenne la main

Tableau XIV. 1 Programme de conversion linéaire

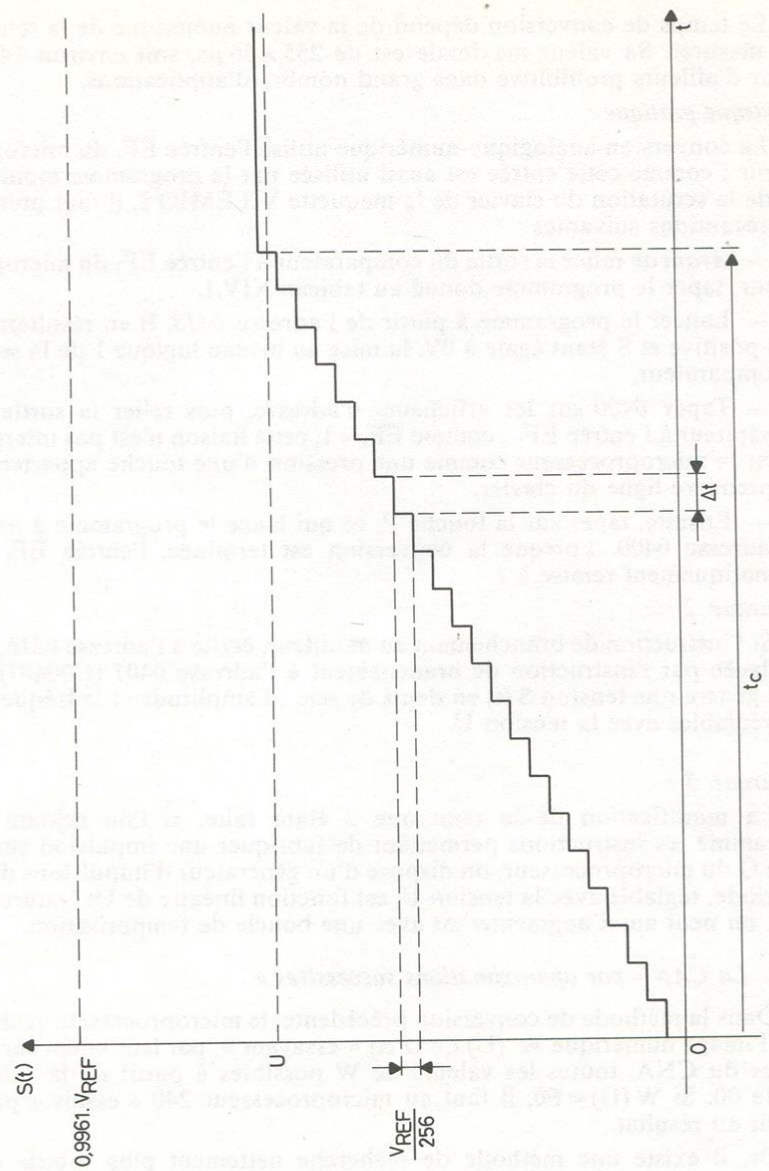


Fig. XIV. 7. Forme de la tension S pendant le déroulement du programme

Le temps de conversion dépend de la valeur numérique de la tension U à mesurer. Sa valeur maximale est de 255×56 µs, soit environ 14 ms, valeur d'ailleurs prohibitive dans grand nombre d'applications.

Remarque pratique :

La conversion analogique-numérique utilise l'entrée $\overline{EF_1}$ du microprocesseur ; comme cette entrée est aussi utilisée par le programme moniteur lors de la scrutation du clavier de la maquette VILÉMIO 3, il faut prendre les précautions suivantes :

— *Avant* de relier la sortie du comparateur à l'entrée $\overline{EF_1}$ du microprocesseur, taper le programme donné au tableau XIV.1.

— Lancer le programme à partir de l'adresse *0413*. Il en résultera, U étant positive et S étant égale à 0V, la mise au niveau logique 1 de la sortie du comparateur.

— Taper *0400* sur les afficheurs d'adresse, puis relier la sortie du comparateur à l'entrée $\overline{EF_1}$: comme $\overline{EF_1} = 1$, cette liaison n'est pas interprétée par le microprocesseur comme une pression d'une touche appartenant à la première ligne du clavier.

— Ensuite, taper sur la touche P, ce qui lance le programme à partir de l'adresse 0400. Lorsque la conversion est terminée, l'entrée $\overline{EF_1}$ est automatiquement remise à 1.

Remarque 2 :

Si l'instruction de branchement au moniteur, écrite à l'adresse 0416, est remplacée par l'instruction de branchement à l'adresse 0407 (C00407), le CNA génère une tension S (t) en dents de scie ; l'amplitude et la fréquence sont réglables avec la tension U.

Remarque 3 :

La modification de la remarque 2 étant faite, si l'on rajoute au programme les instructions permettant de fabriquer une impulsion sur la sortie Q du microprocesseur, on dispose d'un générateur d'impulsions dont la période, réglable avec la tension U, est fonction linéaire de U. Naturellement, on peut aussi augmenter Δt avec une boucle de temporisation.

III. 3. La CAN « par approximations successives »

Dans la méthode de conversion précédente, le microprocesseur recherchait l'image numérique W (U) de U en « essayant », par leur envoi sur les entrées du CNA, toutes les valeurs de W possibles à partir de la valeur initiale 00. Si W (U) = F0, il faut au microprocesseur 240 « essais » pour aboutir au résultat.

Or, il existe une méthode de recherche nettement plus subtile qui permet d'aboutir au résultat en seulement 8 « essais », et ceci quelle que soit la valeur numérique de la tension U à mesurer. Cette méthode est exprimée par l'organigramme de la figure XIV. 8 :

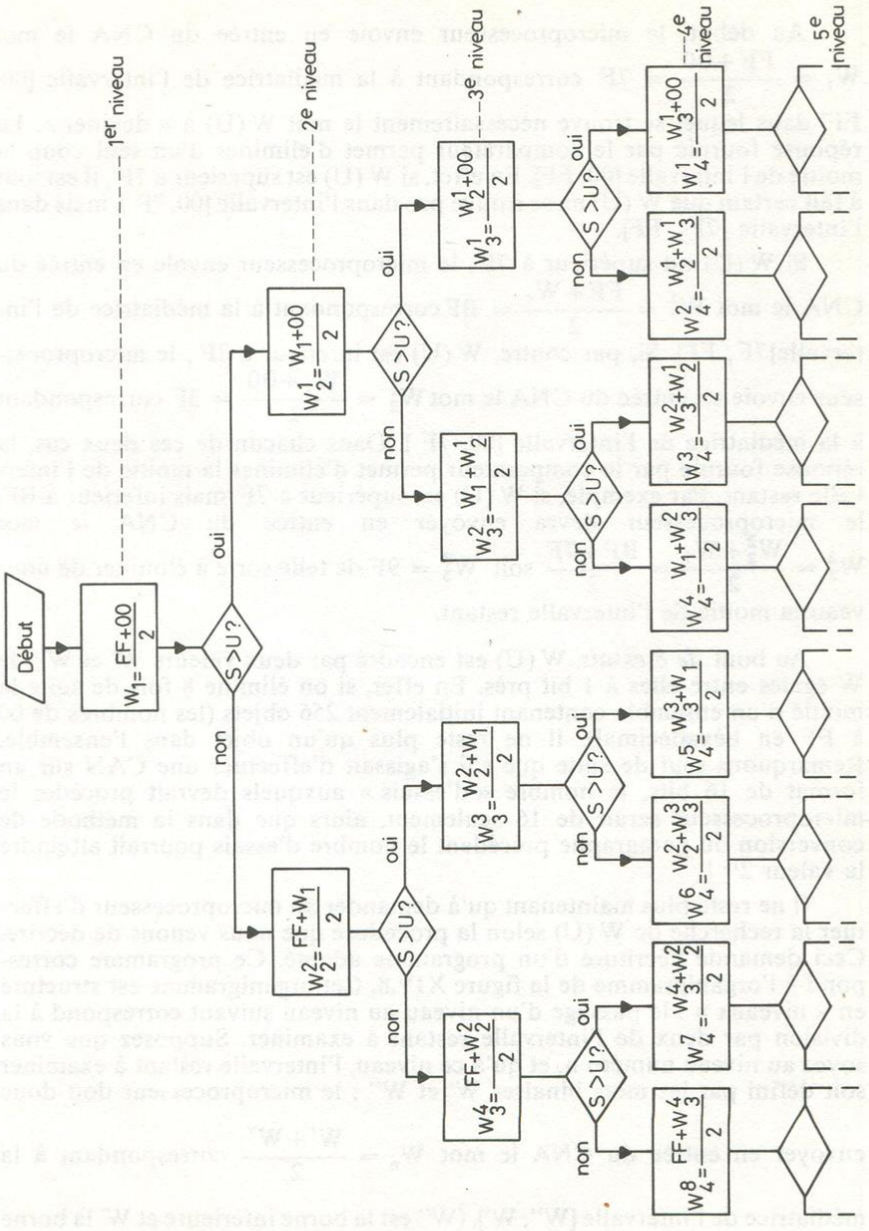


Fig. XIV. 8. Organigramme de « CAN par approximations successives »

Au début, le microprocesseur envoie en entrée du CNA le mot $W_1 = \frac{FF+00}{2} = 7F$ correspondant à la médiatrice de l'intervalle [00, FF] dans lequel se trouve nécessairement le mot W (U) à « deviner ». La réponse fournie par le comparateur permet d'éliminer d'un seul coup la moitié de l'intervalle [00, FF]. En effet, si W (U) est supérieur à 7F , il est tout à fait certain que W (U) ne se trouve pas dans l'intervalle [00, 7F], mais dans l'intervalle]7F , FF].

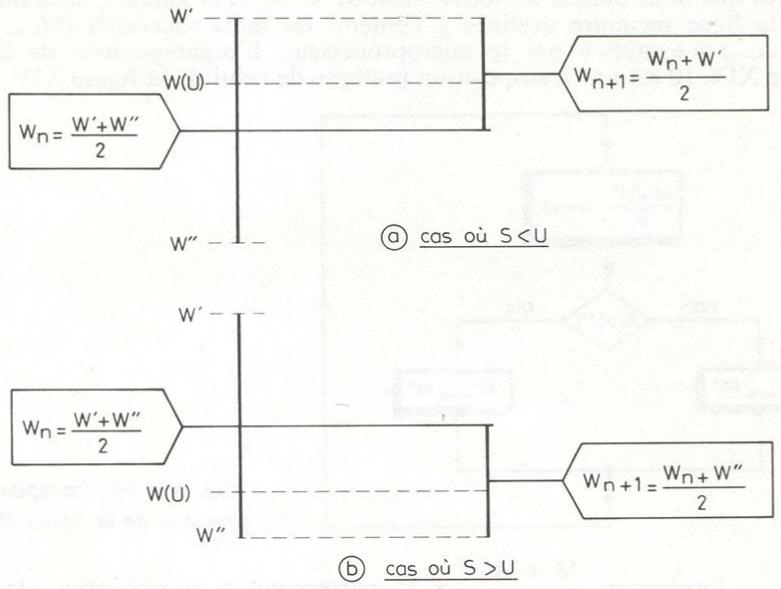
Si W (U) est supérieur à 7F, le microprocesseur envoie en entrée du CNA le mot $W_2^2 = \frac{FF+W_1}{2} = BF$ correspondant à la médiatrice de l'intervalle]7F, FF]. Si, par contre, W (U) est inférieur à 7F , le microprocesseur envoie en entrée du CNA le mot $W_2^1 = \frac{W_1+00}{2} = 3F$ correspondant à la médiatrice de l'intervalle [00, 7F]. Dans chacun de ces deux cas, la réponse fournie par le comparateur permet d'éliminer la moitié de l'intervalle restant. Par exemple, si W (U) est supérieur à 7F mais inférieur à BF, le microprocesseur devra envoyer en entrée du CNA le mot $W_3^3 = \frac{W_2^2+W_1}{2} = \frac{BF+7F}{2}$ soit $W_3^3 = 9F$ de telle sorte à éliminer de nouveau la moitié de l'intervalle restant.

Au bout *de 8 essais,* W (U) est encadré par deux valeurs W' et W'' de W égales entre elles à 1 bit près. En effet, si on élimine 8 fois de suite la moitié d'un ensemble contenant initialement 256 objets (les nombres de 00 à FF en héxadécimal), il ne reste plus qu'un objet dans l'ensemble. Remarquons tout de suite que s'il s'agissait d'effectuer une CAN sur un format de 16 bits, le nombre « d'essais » auxquels devrait procéder le microprocesseur serait de 16 seulement, alors que dans la méthode de conversion du paragraphe précédent le nombre d'essais pourrait atteindre la valeur 2^{16} !

Il ne reste plus maintenant qu'à demander au microprocesseur d'effectuer la recherche de W (U) selon la procédure que nous venons de décrire. Ceci demande l'écriture d'un programme adapté. Ce programme correspond à l'organigramme de la figure XIV.8. Cet organigramme est structuré en « niveaux » : le passage d'un niveau au niveau suivant correspond à la division par deux de l'intervalle restant à examiner. Supposez que vous soyez au niveau numéro n, et qu'à ce niveau, l'intervalle restant à examiner soit défini par les mots binaires W' et W'' ; le microprocesseur doit donc envoyer en entrée du CNA le mot $W_n = \frac{W'+W''}{2}$ correspondant à la médiatrice de l'intervalle [W'', W']. (W'' est la borne inférieure et W' la borne supérieure).

Selon la réponse du comparateur à cet « essai », le microprocesseur passera au niveau suivant de numéro $n+1$ en envoyant en entrée du CNA un mot W_{n+1} différent selon que $W(U)$ est supérieur ou inférieur à W_n :

— si $W(U) > W_n$ (fig. XIV. 9-a), l'intervalle suivant à examiner est $[W_n, W']$ et W_{n+1} doit être égal à $\frac{W_n + W'}{2}$.



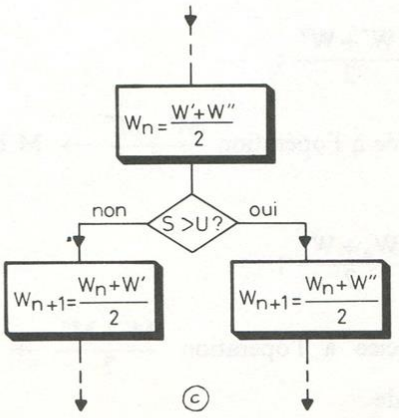


Fig. XIV. 9. Décomposition des « approximations successives »

— si $W(U) < W_n$ (fig. XIV. 9-b), l'intervalle suivant à examiner est [W'', W_n] et W_{n+1} doit être égal à $\frac{W_n + W''}{2}$.

L'organigramme de la figure XIV. 9-c exprime de quelle façon il faut passer du niveau n au niveau n + 1.

Supposez maintenant que W' et W'' sont dans deux lignes de mémoire RAM que nous notons respectivement M' et M''. Par ailleurs, nous notons M la ligne mémoire destinée à contenir les mots successifs (W_1..., W_n, W_{n+1},...) « essayés » par le microprocesseur. L'organigramme de la figure XIV. 10 est une transposition pratique de celui de la figure XIV. 9-c :

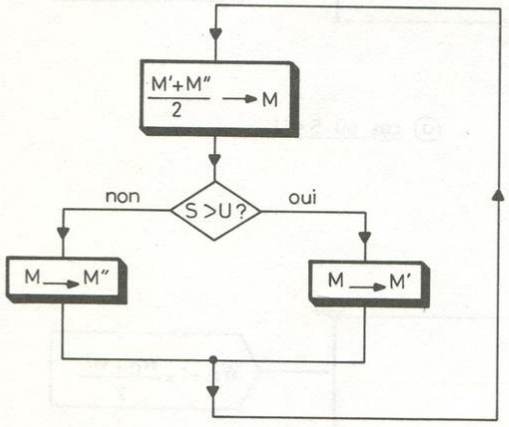


Fig. XIV. 10. Transposition pratique de la figure XIV. 9.

— l'opération $\frac{M' + M''}{2} \rightarrow M$ correspond à la réalisation de

$$W_n = \frac{W' + W''}{2} ;$$

— l'opération $M \rightarrow M'$, associée à l'opération $\frac{M' + M''}{2} \rightarrow M$ correspond à la réalisation de

$$W_{n+1} = \frac{W_n + W''}{2} ;$$

— l'opération $M \rightarrow M''$, associée à l'opération $\frac{M' + M''}{2} \rightarrow M$, correspond à la réalisation de

$$W_{n+1} = \frac{W_n + W'}{2}.$$

Enfin, l'organigramme de la figure XIV. 11 représente le détail de la procédure de CAN par cette méthode. Cet organigramme contient l'initialisation de M' et M'' respectivement à FF et 00, c'est-à-dire la définition de l'intervalle initial [00, FF]. Par ailleurs, le calcul de $\frac{M'+M''}{2}$ s'effectue en calculant d'abord $\frac{M'-M''}{2}$ puis en rajoutant M'' ; ceci est justifié par le

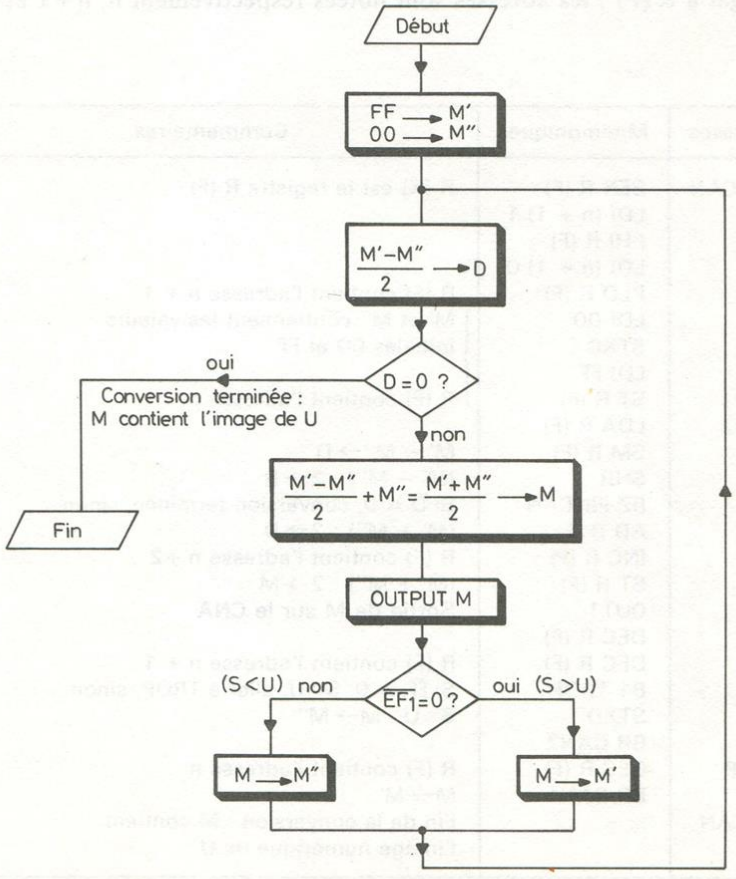


Fig. XIV. 11. Méthode détaillée de CAN par approximations

fait que nous avons besoin de connaître $\frac{M'-M''}{2}$ pour décider si la conversion est terminée. En effet, la conversion est terminée lorsque l'écart entre M' et M" est de 1 bit, c'est-à-dire lorsque $\frac{M'-M''}{2}$ (la division par 2 étant obtenue par un décalage à droite) est égal à 0.

Le tableau XIV. 2 donne, en langage assembleur, le programme correspondant à l'organigramme de la figure XIV. 11. Les trois lignes mémoire RAM notées M', M" et M sont pointées par le registre R (X) que nous avons pris égal à R (F) ; les adresses sont notées respectivement n, n+1 et n+2

Adresses	Mnémoniques	Commentaires
DEBCAN	SEX R (F)	R (X) est le registre R (F)
	LDI (n + 1).1	
	PHI R (F)	
	LDI (n + 1).0	
	PLO R (F)	R (F) contient l'adresse n + 1
	LDI 00	M' et M'' contiennent les valeurs
	STXD	initiales 00 et FF
	LDI FF	
CAN1	ST R (F)	R (F) contient l'adresse n
CAN2	LDA R (F)	
	SM R (F)	M' − M''→D
	SHR	(M' − M'') : 2→D
	BZ FINCAN	Si D = 0, conversion terminée, sinon...
	AD R (F)	(M' + M'') : 2→D
	INC R (F)	R (F) contient l'adresse n +2
	ST R (F)	(M' + M'') : 2→M
	OUT1	Sortie de M sur le CNA
	DEC R (F)	
	DEC R (F)	R (F) contient l'adresse n + 1
	B1 TROP	Si $\overline{EF}_1$ = 0, S>U, aller à TROP, sinon...
	STXD	S<U : M→M''
	BR CAN2	
TROP	DEC R (F)	R (F) contient l'adresse n
	BR CAN1	M→M'
FINCAN	...	Fin de la conversion : M contient l'image numérique de U

Tableau XIV. 2. Programme en assembleur correspondant à la figure XIV.11.

(fig. XIV. 12). Nous utiliserons ce programme de conversion dans le paragraphe suivant. Cependant, nous pouvons dès maintenant prévoir clairement comment se comporte ce programme pendant la recherche de l'image numérique W (U) de U :

Supposez que la tension U à mesurer soit de *6,81 V* et la tension de référence réglée à 10 V. Le tableau XIV. 3 indique les valeurs de M', M'', (M' − M''): 2 et M à chaque fois qu'elles sont calculées. Les 8 valeurs successives de S fournies par 8 exécutions successives de l'instruction OUTPUT sont également données. Lorsque la conversion est terminée, *S = 6,80 V*. Ainsi, l'image numérique de U est, à l'erreur systématique inévitable près, le nombre binaire *W (U) = AE*. La figure XIV. 13 représente l'évolution de S pendant la conversion ; il s'agit globalement d'une fonction oscillatoire amortie exponentiellement vers la valeur U à mesurer.

Le temps de conversion t_c est égal à 8 fois Δt, Δt étant l'intervalle de temps entre deux éxécutions successives de l'instruction OUTPUT. L'examen du tableau XIV. 2 montre que Δt vaut à peu près 110 µs pour une

M'	FF	FF	BF	BF	AF	AF	AF	AF	AF
M''	00	7F	7F	9F	9F	A7	AB	AD	AE
$\frac{M' - M''}{2}$	7F	40	20	10	08	04	02	01	00
M	7F	BF	9F	AF	A7	AB	AD	AE	AE
$\frac{S}{V_{REF}}$	0,496	0,746	0,621	0,684	0,652	0,668	0,676	0,680	0,680
S (en V.)	4,96	7,46	6,21	6,84	6,52	6,68	6,76	6,80	6,80

Tableau XIV.3.

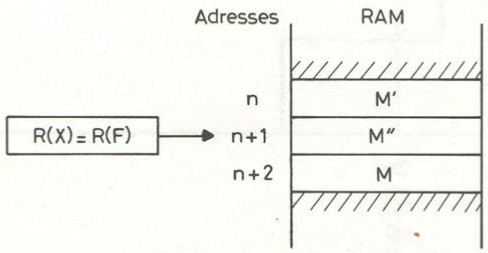


Fig. XIV. 12. Position de M, M', M'' dans la RAM

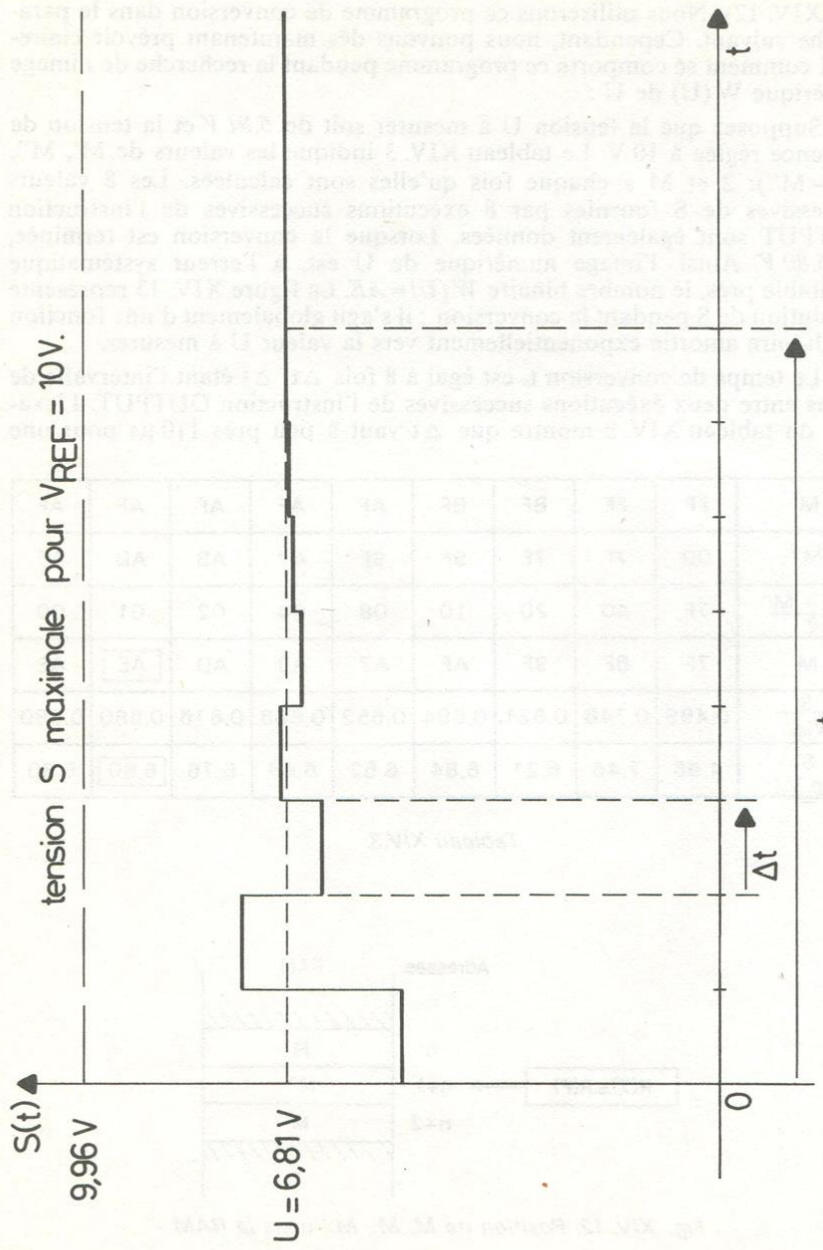


Fig. XIV. 13. Evolution de S pendant la conversion

fréquence d'horloge de 2 MHz, ce qui donne un temps de conversion de l'ordre de *0,9 ms.* Cette valeur est à comparer au chiffre de 14 ms trouvé lors de l'étude de la CAN linéaire.

Remarque 1

Si l'on boucle le programme de conversion sur lui-même, on remarque que le seul but du microprocesseur est de rendre les tensions d'entrée U et S du comparateur constamment égales. Ceci peut-être exploité à la mise en œuvre des « asservissements ». Voyez par exemple la figure XIV. 14 : le CNA commande la vitesse Ω d'un moteur à courant continu. L'arbre du moteur est couplé à une génératrice tachymétrique fournissant une tension $k \times \Omega$ proportionnelle à la vitesse de rotation Ω. Si une perturbation extérieure au système (une charge variable par exemple) vient à modifier temporairement la vitesse du moteur, le microprocesseur réagit automatiquement de telle sorte que cette variation de vitesse soit compensée. Autrement dit, la vitesse Ω est imposée par la tension U selon la relation $U = k \times \Omega$: Ω est asservie à U.

Remarque 2

Pendant toute la durée de la conversion, il est souhaitable que la tension U à mesurer ne varie pas, ou en tous cas ne varie pas plus que la résolution de 39 mV dont nous avons déjà parlé. Autrement dit, la vitesse de variation de U ne doit pas dépasser 39 mV/0,9 ms soit environ 40 mV par ms.

IV. Application à la mesure d'une température

IV. 1. Transformer une température en tension

Nous désirons mesurer une température T, ou en tout cas une variation de température ΔT par rapport à une température de référence T_0 connue. Comme nous savons déjà convertir une tension U en son image numérique W (U), il apparaît qu'il nous reste le problème de la transformation de la température T en une tension U. Pour ce faire, il nous faut un *capteur.*

La figure XIV. 15-a présente notre choix. Il s'agit du capteur AD 590 M (ANALOG DEVICES) qui présente la propriété de fournir un courant I proportionnel à la température *absolue* T selon la relation $I = k.\ T$; la valeur de k est de *1 µA par degré.* Cette relation est vraie dans la gamme de températures allant de -55 °C à $+150$ °C (c'est-à-dire de 218 °K à 423 °K) avec cependant les restrictions suivantes :

a) En réalité, pour une température absolue T donnée, le courant I n'est pas rigoureusement égal à kT, mais à $k\ (T + T_c)$. T_c représente ce qu'on appelle *l'erreur de calibrage* et vaut au maximum 0,5 degré en valeur absolue. Ceci peut-être compensé par un calibrage électrique.

b) Par ailleurs, la relation $I = kT$ n'est pas rigoureusement linéaire. Cela veut dire que, pour un courant I mesuré, la température n'est pas rigoureusement égale à I/k, mais égale à cette valeur à une certaine *erreur de non-linéarité* près. Cette erreur est au maximum de 0,3° en valeur absolue.

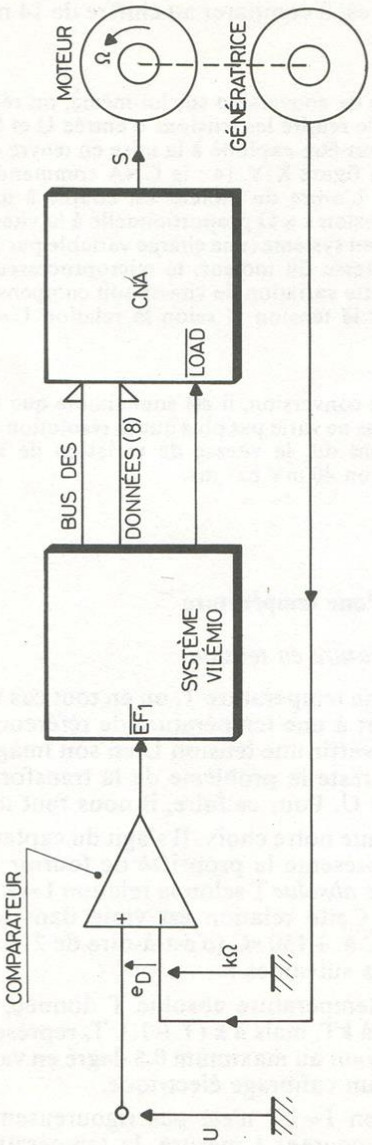


Fig. XIV. 14. Commande de vitesse

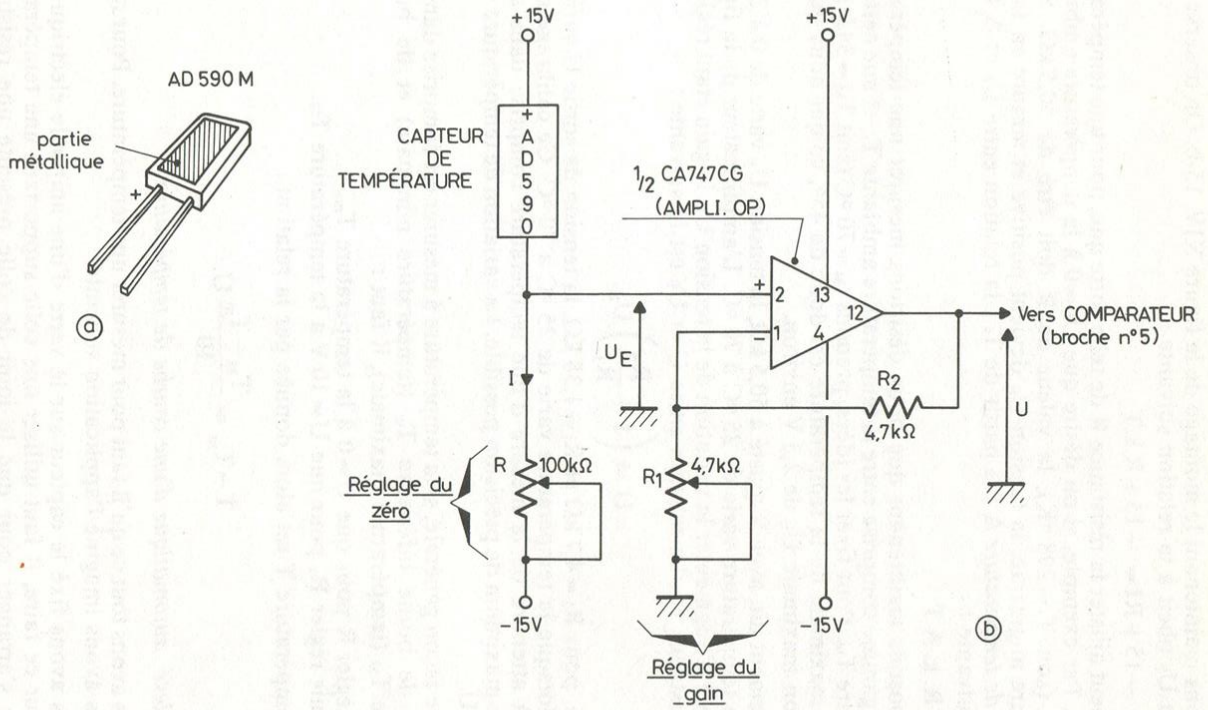


Fig. XIV. 15. Capteur de température et montage d'utilisation

Voyons maintenant le montage de la figure XIV. 15-b. On observe que la tension U_E obéit à la relation suivante :

$$U_E = -15 + RI = -15 + R.k.T$$

On peut ajuster la résistance R de telle sorte que, pour une température T, $U_E = 0$. Par exemple, si on désire que $U_E = 0$ à la température ambiante $t_a = 25\ ^oC$ (soit $T_a = 298\ ^oK$), la valeur de R doit être de 50,5 kΩ. Si la température augmente, la tension U_E devient positive et mesure en fait *la variation de température* ΔT à partir de T_a ; la relation entre U_E et ΔT est alors la suivante :

$$U_E = R.\,k.\,\Delta T$$

Supposons maintenant que nous désirions mesurer une température dans une gamme comprise entre la température ambiante T_a, et une certaine température T_M. Pour fixer les idées, prenons $t_M = 70\ ^oC$ (soit $T_M = 343^o$) ; la variation maximale de la température est donc de 45°, ce qui nous donne une tension maximale U_E de 2,3 V environ.

Autrement dit, avec R ajusté à 50,5 kΩ, la tension U_E varie de 0 à 2,3 V lorsque la température varie de 25 °C à 70 °C. L'amplificateur de la figure 15-b permet d'augmenter la variation de la tension U_E, le gain étant réglable par la résistance R_1. La relation entre U et U_E est la suivante :

$$U = \left(1 + \frac{R_2}{R_1}\right) U_E.$$

Ainsi, pour $R_2 = 4{,}7$ kΩ et $R_1 = 1{,}38$ kΩ, la tension de sortie U varie de 0 à 10 V lorsque la température varie de 25 °C à 70 °C. Ce calibrage nous autorise à attendre de la mesure d'une température comprise dans cette gamme, le maximum de précision possible. La variation de température vaut $\Delta T = 4{,}5\ U$.

D'une façon générale, si la température à mesurer est comprise dans un intervalle de borne inférieure T_m (température minimale) et de borne supérieure T_M (température maximale), il faut :

— Régler R pour que $U = 0$ à la température T_m.
— Puis régler R_1 pour que $U = 10$ V à la température T_M.

La température T est alors donnée par la relation :

$$T - T_m = \frac{T_M - T_m}{10} U.$$

IV. 2. Relevé automatique d'une courbe de température

Nous avons tout ce qu'il faut pour mesurer une température. Pour notre part, nous avons imaginé l'application suivante.

Nous avons fixé le capteur sur le verre d'une ampoule électrique de 60 W. Pour ce faire, il faut utiliser une colle supportant une température élevée, et s'arranger pour que le joint de colle présente une résistance thermique très faible. Sinon, on se débrouille autrement. Si l'on branche ou

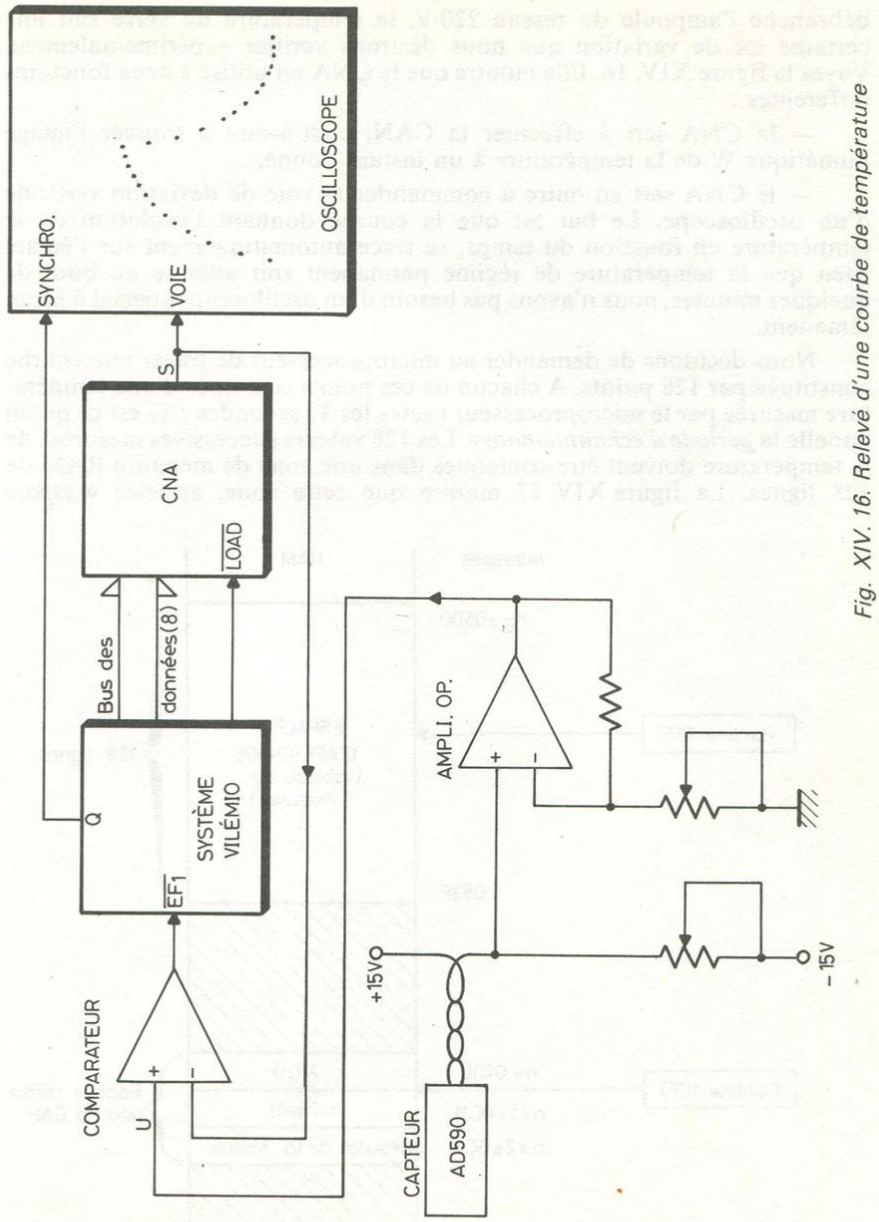


Fig. XIV. 16. Relevé d'une courbe de température

débranche l'ampoule du réseau 220 V, la température du verre suit une certaine loi de variation que nous désirons vérifier expérimentalement. Voyez la figure XIV. 16. Elle montre que le CNA est utilisé à deux fonctions différentes :

— le CNA sert à effectuer la CAN, c'est-à-dire à trouver l'image numérique W de la température à un instant donné.

— le CNA sert en outre à commander la voie de déviation verticale d'un oscilloscope. Le but est que la courbe donnant l'évolution de la température en fonction du temps, se trace automatiquement sur l'écran. Bien que la température de régime permanent soit atteinte au bout de quelques minutes, nous n'avons pas besoin d'un oscilloscope spécial à écran rémanent.

Nous décidons de demander au microprocesseur de tracer une courbe constituée par 128 points. A chacun de ces points correspond une température mesurée par le microprocesseur toutes les T_E secondes : T_E est ce qu'on appelle la *période d'échantillonnage*. Les 128 valeurs successives mesurées de la température doivent être contenues dans une zone de mémoire RAM de 128 lignes. La figure XIV. 17 montre que cette zone, appelée « *espace*

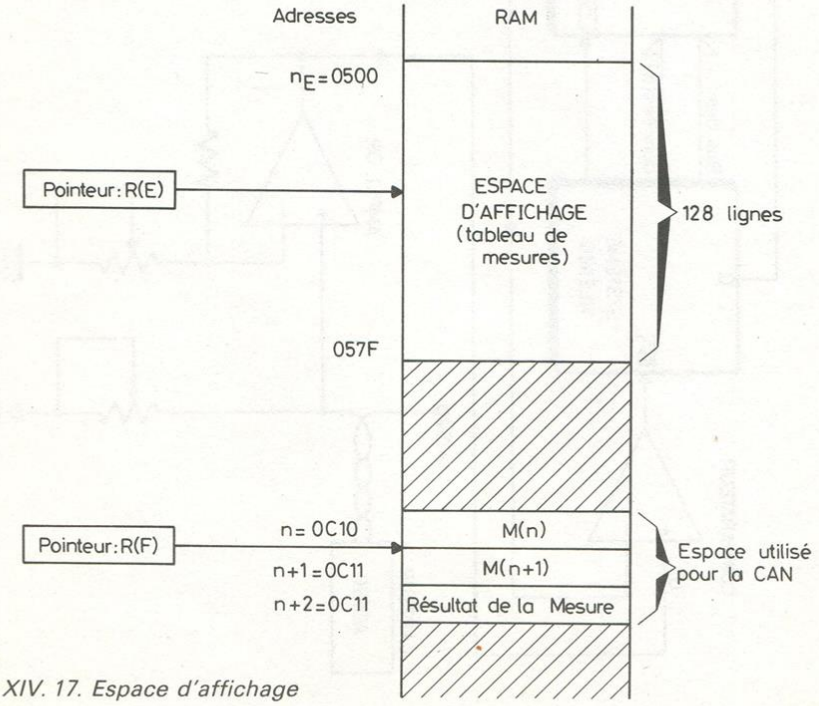


Fig. XIV. 17. Espace d'affichage

d'affichage », ou « *tableau de mesures* », est située aux adresses de 0500 à 057F ; le registre R (E) pointe cette zone.

Compte tenu de l'ordre de grandeur en durée du phénomène à observer, nous avons choisi une période d'échantillonnage T_E de l'ordre de 5s. Autrement dit, nous demandons au microprocesseur d'effectuer une mesure de température toutes les 5s, et de stocker le résultat au bon endroit dans le « tableau de mesures ». 128 mesures devant être faites, la collecte des données prend environ 10 minutes.

Au fur et à mesure de l'évolution du phénomène à observer, l'« espace d'affichage » se remplit progressivement en s'enrichissant de données supplémentaires. Pour que cette évolution soit lisible « en continu » sur l'écran de l'oscilloscope, il nous faut demander au microprocesseur d'afficher constamment la zone RAM contenant le résultat des mesures. Pour ce faire, il convient de :

a) exécuter une série de 128 instructions OUTPUT, de telle sorte que les 128 mots inscrits dans la mémoire à partir de l'adresse 0500 soient affichés sur l'écran. C'est la base de temps interne de l'oscilloscope qui est utilisée ; elles est synchronisée par un signal généré par la sortie Q du microprocesseur.

b) revenir en a) de telle sorte à rafraîchir l'affichage.

L'organigramme général de ce banc de mesure automatique est donné figure XIV. 18. Au début, le « tableau de mesures » est mis à zéro, ce qui correspond au fait que l'on commence la manipulation, et que la température du verre de l'ampoule électrique est à sa valeur minimale (température ambiante). Ensuite, on allume l'ampoule et on fait dérouler le programme ; que se passe t-il ?

Tout d'abord, un *compteur de mesures*, le demi-registre R (C).0, est initialisé à FF. Ce compteur indique le numéro d'ordre de la mesure effectuée : à la 1re mesure R (C).0 contient 00, à la 2e mesure R (C).0 contient 01, à la 3e 02, etc.

Ensuite, un *compteur de la période d'échantillonnage*, le registre R (D), est initialisé à une valeur telle que T_E soit de l'ordre de 5s.

Puis a lieu pendant la durée T_E une suite d'affichages du « tableau de mesures ». Au début de la manipulation, la figure tracée sur l'écran de l'oscilloscope est une ligne horizontale continue représentant l'axe des temps.

Lorsque la durée T_E est écoulée, le registre compteur de mesures R (C).0 est incrémenté, puis une mesure (CAN) est effectuée. Le résultat est stocké dans le tableau de mesures à une adresse définie par le registre R (C).0.

Voilà. Si vous essayez le programme donné dans le tableau XIV. 4, vous verrez se décrire sur l'écran de l'oscilloscope, point par point, une fonction exponentielle croissant vers une valeur limite. Cette fonction représente la

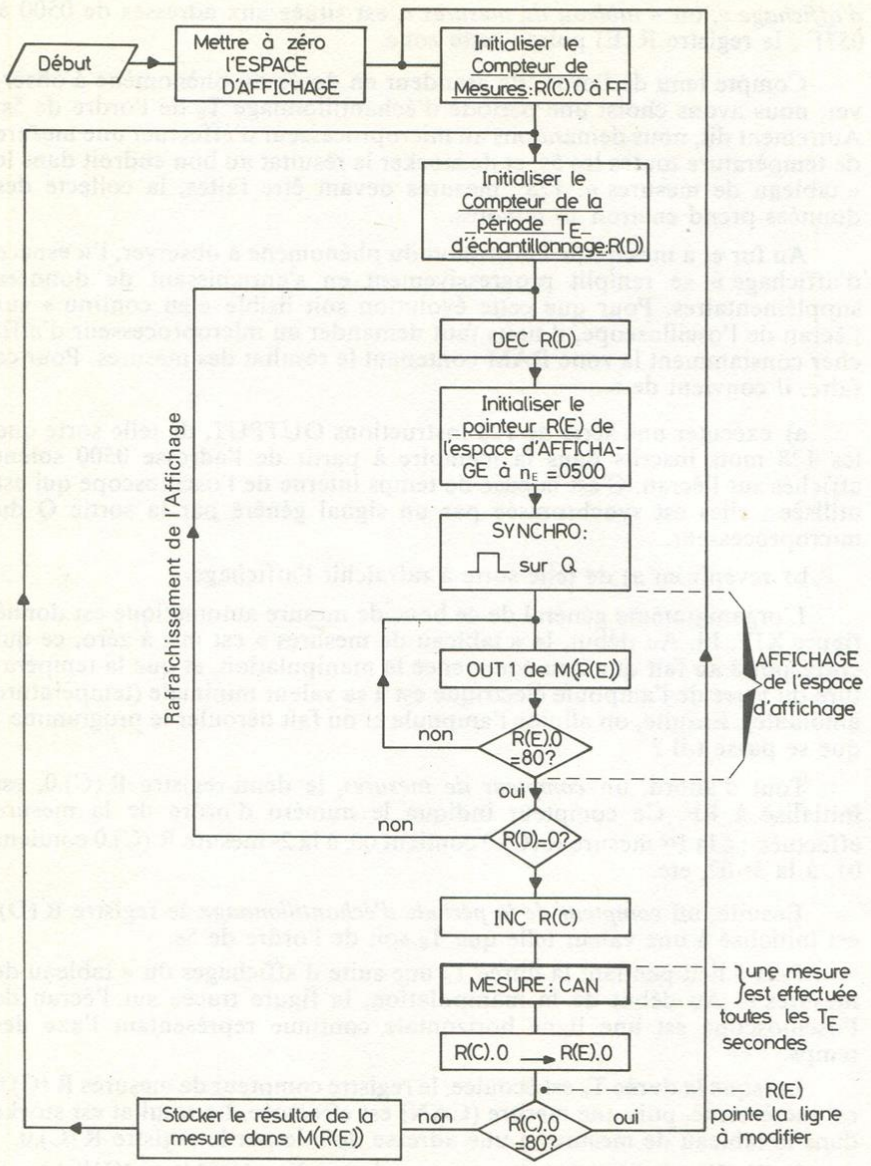


Fig. XIV. 18. Organigramme d'un banc de mesure automatique

Adresses	Codes et opérandes	Adresses	Mnémoniques	Commentaires
			Mettre à zéro L'ESPACE D'AFFICHAGE	
0400	F8 05	DÉBUT	LDI 05	
2	BE		PHI R (E)	
3	F8 7F		LDI 7F	
5	AE		PLO R (E)	R (E) contient 057F
6	EE		SEX R (E)	R (X) = R (E)
7	8E	ST00	GLO R (E)	
8	FF FF		SMI FF	Si R (E) contient 04FF, soit si
A	32 11		BZ INIMES	R (E).0 contient FF, terminé...
C	F8 00		LDI 00	sinon...
E	73		STXD	stocker 00 dans M (R (E))
F	30 07		BR ST00	
			Initialiser le compteur de mesures	
0411	F8 FF	INIMES	LDI FF	
3	AC		PLO R (C)	R (C).0 contient FF
			Initialiser le compteur de la période T_E d'échantillonnage	
0414	F8 07	INIECH	LDI 07	
6	BD		PHI R (D)	
7	F8 00		LDI 00	
9	AD		PLO R (D)	R (D) contient 0700
			Affichage pendant la période T_E	
041A	2D	AFFT	DEC R (D)	
B	F8 05		LDI 05	
D	BE		PHI R (E)	
E	F8 00		LDI 00	
20	AE		PLO R (E)	R (E) contient 0500 = n_E
1	7B		SEQ	Synchronisation de
2	7A		REQ	l'oscilloscope
3	EE		SEX R (E)	R (X) = R (E)
4	61	AFF	OUT1	} Affichage
5	8E		GLO R (E)	
6	FF 80		SMI 80	
8	3A 24		BNZ AFF	
A	8D		GLO R (D)	} Test de R (D)
B	3A 1A		BNZ AFFT	
D	9D		GHI R (D)	
E	3A 1A		BNZ AFFT	
30	1C		INC R (C)	

Mesure : C.A.N.

0431	EF	DEBCAN	SEX R (F)	R (X) = R (F)
2	F8 0C		LDI (n + 1).1	(n + 1).1 = 0C
4	BF		PHI R (F)	
5	F8 11		LDI (n + 1.)0	(n + 1).0 = 11
7	AF		PLO R (F)	R (F) contient n +1 = 0C11
8	F8 00		LDI 00	
A	73		STXD	M (n + 1) contient 00
B	F8 FF		LDI FF	
D	5F	CAN1	ST R (F)	M (n) contient initialement FF
E	4F	CAN2	LDA R (F)	
F	F7		SM R (F)	M (n) − M (n + 1)→D
40	F6		SHR	D : 2→D
1	32 51		BZ FINCAN	si D = 0.conversion terminée, sinon
3	F4		AD R (F)	...
4	60		INC R (F)	...
5	5F		ST R (F)	{ [M (n) + M (n + 1)] : 2 { →M (n + 2)
6	61		OUT1	Sortie de M (n + 2) sur le CNA
7	2F		DEC R (F)	
8	2F		DEC R (F)	
9	34 4E		B1 TROP	{ Si $\overline{EF_1}$ = 0,S > U aller à TROP, { sinon
B	73		STXD	S<U : M (n + 2)→ M (n + 1)
C	30 3E	TROP	BR CAN2	
E	2F		DEC R (F)	
F	30 3D		BR CAN1	M (n + 2)→M (n)

Stockage du résultat de la mesure

0451	F8 12	FINCAN	LDI (n + 2).0	(n + 2).0 = 12
3	AF		PLO R (F)	R (F) contient 0C12
4	8C		GLO R (C)	
5	AE		PLO R (E)	R (C).0→R (E).0
6	FF 80		SMI 80	si R (E).0 = 80. 128 mesures ont
8	32 11		BZ INIMES	été effectuées :
A	0F		LD R (F)	réinitialiser R (C)
B	5E		ST R (E)	Résultat de la mesure stocké
045C	30 14		BR INIECH	

Tableau XIV. 4. Programme du banc de mesure

montée en température du verre de l'ampoule électrique. On peut arrêter d'alimenter l'ampoule pendant que le programme tourne ; c'est alors une fonction exponentielle décroissante qui est lentement tracée point par point.

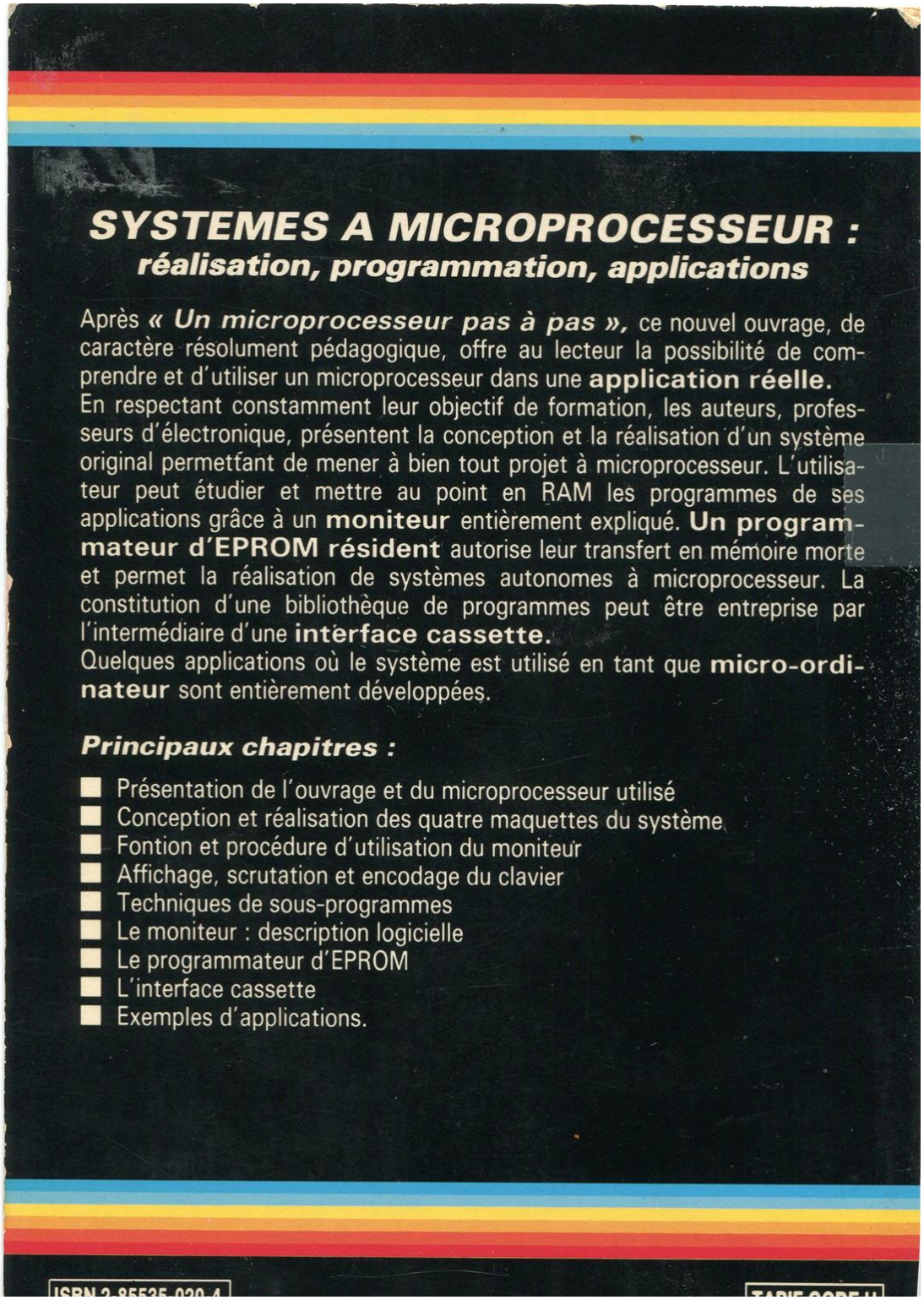
Remarque 1

Pour une meilleure définition de la courbe tracée, rien n'empêche d'utiliser un tableau de mesures de 5×128 lignes par exemple, et une période d'échantillonnage T_E de l'ordre de la seconde. La définition est pratiquement limitée par la taille mémoire disponible.

Remarque 2

Lorsque 128 mesures ont été effectuées, le microprocesseur continue naturellement les mesures (voir l'organigramme) mais stocke le résultat de la 129e mesure à l'endroit où se trouvait stocké le résultat de la 1re, et ainsi de suite. L'illusion obtenue sur l'écran de l'oscilloscope est alors celle d'un papier graphique qui se déplace très lentement de gauche à droite.

Achevé d'imprimer sur les presses de l'Imprimerie Marcel Bon 70001 Vesoul
Dépôt légal : Avril 1983 - N° d'éditeur : 375 - N° d'imprimeur : 2665


SYSTEMES A MICROPROCESSEUR :
réalisation, programmation, applications
Après « Un microprocesseur pas à pas », ce nouvel ouvrage, de caractère résolument pédagogique, offre au lecteur la possibilité de comprendre et d'utiliser un microprocesseur dans une application réelle.
En respectant constamment leur objectif de formation, les auteurs, professeurs d'électronique, présentent la conception et la réalisation d'un système original permettant de mener à bien tout projet à microprocesseur. L'utilisateur peut étudier et mettre au point en RAM les programmes de ses applications grâce à un moniteur entièrement expliqué. Un programmateur d'EPROM résident autorise leur transfert en mémoire morte et permet la réalisation de systèmes autonomes à microprocesseur. La constitution d'une bibliothèque de programmes peut être entreprise par l'intermédiaire d'une interface cassette.
Quelques applications où le système est utilisé en tant que micro-ordinateur sont entièrement développées.
Principaux chapitres :
Présentation de l'ouvrage et du microprocesseur utilisé
Conception et réalisation des quatre maquettes du système
Fontion et procédure d'utilisation du moniteur
Affichage, scrutation et encodage du clavier
Techniques de sous-programmes
Le moniteur : description logicielle
Le programmateur d'EPROM
L'interface cassette
Exemples d'applications.

END