

André VILLARD

Michel MIAUX

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

Un microprocesseur pas à pas

Diffusion :

ÉDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES

2 à 12, rue de Bellevue, 75940 PARIS CEDEX 19

(1981)

SOMMAIRE

Première partie Structure et fonctionnement des processeurs

Chapitre 1 - Les mémoires

I.	Organisation d'une mémoire	13
II.	L'accès à la mémoire	15
III.	Les mémoires RAM	18
IV.	La sélection de boîtier	21
V.	Les différents types de mémoires	24

Chapitre 2 - Automate programmable

I.	Description et fonctionnement d'un automate programmable	29
II.	Exemples d'utilisation	31
III.	Assimilation	37

Chapitre 3 - Automate programmable composé

I.	Description et fonctionnement du système	53
II.	Exemples d'utilisation	59
III.	Assimilation	68

Chapitre 4 - Notion de processeur

I.	Elaboration d'un processeur élémentaire	73
II.	Programmation du processeur élémentaire	77
III.	Le microprocesseur	84

Chapitre 5 - Structure du microprocesseur Cosmac CDP 1802

I. L'ensemble décodeur d'instruction et bloc de contrôle	89
II. Le bus des données	91
III. Les registres R et les éléments associés	91
IV. Les différents rôles des registres R. Les registres pointeurs P, X, N ...	92
V. L'ALU, le registre D et le registre DF	94
VI. Le registre T	95
VII. Les registres d'instruction I et N	95
VIII. Les bascules Q et IE	95

Chapitre 6 - Les instructions du microprocesseur CDP 1802

I. Considérations générales	97
II. Jeu d'instructions du microprocesseur CDP 1802	99

Deuxième partie

Conception et réalisation d'une maquette d'aide à l'étude de la mise en œuvre et de la programmation d'un microprocesseur

<i>Introduction</i>	105
---------------------------	-----

Chapitre 7 : Conception d'une maquette d'aide à l'étude de la mise en œuvre et de la programmation d'un μ P :

I. L'horloge	109
II. Le couplage du microprocesseur à la mémoire	112
III. L'accès direct à la mémoire	121
IV. Les « interruptions » de programme ou, « comment dialoguer avec le système »	126
V. Les signaux d'état	127
VI. Le contrôle du microprocesseur	130
VII. Les communications avec le monde extérieur	131
VIII. Le mode de fonctionnement « LOAD » du μ P CDP 1802	136

Chapitre 8 - Réalisation pratique de la maquette A

I. Présentation du matériel utilisé	145
II. Liste du matériel nécessaire à la réalisation de la maquette A complète	152
III. Réalisation pratique de la maquette A	153

Chapitre 9 - Réalisation pratique de la maquette B

I. Présentation et rappel du rôle du matériel utilisé	157
II. Liste du matériel nécessaire à la réalisation de la maquette B	161
III. Réalisation pratique de la maquette B	162

Troisième partie

Etude des principes fondamentaux de la mise en œuvre et de la programmation du microprocesseur

Chapitre 10 - Etude en pas-à-pas de l'enregistrement et du déroulement d'un programme élémentaire.

I. Position du problème	163
II. Enregistrement du programme	164
III. Déroulement du programme	168
IV. Conclusion et remarques	174
V. Résumé des opérations et observations	174

Chapitre 11 - Le branchement inconditionnel (Etude en pas-à-pas)

I. Position du problème	177
II. Enregistrement du programme	178
III. Déroulement du programme	179

Chapitre 12 - Le branchement conditionnel

I. Position d'un problème	185
II. Le branchement conditionnel	186
III. Utilisation du branchement conditionnel : le temporisateur	187
IV. Basculer astable dont la durée des états haut et bas est grande devant un cycle	191
V. Assimilation : exercices de programmation	193

Chapitre 13 - Les sous-programmes

I. Mise en évidence de l'utilité des sous-programmes	201
II. Mise en œuvre d'un sous-programme	203
III. Espace de données. Utilisation d'un pointeur de données	208
IV. Elaboration du programme de l'alarme	209

Chapitre 14 - Entrée et sortie de données	
I. Entrée d'une donnée	213
II. Sortie d'une donnée	219
III. Exemples de programmation	222

Chapitre 15 - L'interruption par le canal $\overline{\text{DMA-IN}}$	
I. Position d'un problème	235
II. Mise en œuvre de l'interruption par le canal $\overline{\text{DMA-IN}}$	236
III. Solution proposée au problème énoncé	237
IV. Autre exemple d'interruption par le canal $\overline{\text{DMA-IN}}$	241
V. Note complémentaire	246

Chapitre 16 - L'interruption par le canal $\overline{\text{INTERRUPT}}$	
I. Position d'un problème	249
II. Mise en œuvre de l'interruption par le canal $\overline{\text{INTERRUPT}}$	250
III. Solution proposée au problème énoncé	254
IV. Autre exemple d'interruption par le canal $\overline{\text{INTERRUPT}}$	258
V. Application de l'interruption par le canal $\overline{\text{INTERRUPT}}$: calculatrice 4 opérations	260

Quatrième partie Interfaçage et applications

Chapitre 17 - L'introduction de données	
I. La sélection de fonctions	267
II. Scrutation et décodage d'un clavier	270

Chapitre 18 - L'affichage numérique	
I. La commande d'un afficheur numérique	283
II. L'affichage numérique multiplexé	287
III. Application au comptage	291
IV. Application à la chronométrie	299
V. Interruption de programme et précision	304
IV. Application à la mesure des vitesses de rotation	309
VII. Application aux systèmes programmables	314

Chapitre 19 - La conversion numérique ↔ analogique

I. La conversion numérique → analogique	327
II. La conversion analogique → numérique	337

Chapitre 20 - Les matériels utilisés

I. Eléments de la logique combinatoire	345
II. Eléments de la logique séquentielle	350

INTRODUCTION

Le monde de l'électronique et celui de l'informatique sont traditionnellement séparés par un fossé d'incompréhension : cédant à un a-priori commun, on dresse une barrière entre les problèmes d'ordre purement matériel (structure et fonctionnement des composants électroniques, synthèse des systèmes considérés comme le résultat de leurs interconnexions et les problèmes d'ordre purement logiciel (programmation des systèmes en vue de l'exécution automatique d'une tâche déterminée). Au contraire, cet ouvrage se propose d'établir une liaison continue entre ces deux mondes, bien que plus particulièrement destiné à des électroniciens (ou futurs électroniciens) désireux de comprendre le fonctionnement d'un microprocesseur qu'ils puissent insérer, avec leurs moyens propres, dans un système automatique obéissant à une fonction déterminée. Il convient en effet de ne pas perdre de vue que le microprocesseur est avant tout un composant électronique dont l'organisation interne réunit un certain nombre de matériaux de base, qui sont connus de tout électronicien moyen quant à la fonction et quant à la structure. (Si nécessaire, le lecteur se reportera au chapitre 20).

1 →

2 →

La première partie de l'ouvrage montre pas-à-pas comment construire un (micro) processeur à partir de ces matériaux de base : on présente tout d'abord un automate programmable (automate A) destiné à la commande de divers engins électroniques, qui amène rapidement le lecteur à la notion de fonction programmée ; puis un deuxième automate (automate B) est couplé à l'automate A. L'ensemble automate A - automate B constitue un automate programmable composé, destiné à son tour à la commande de divers systèmes électroniques (Compteurs - Registres - Unité arithmétique et logique - Mémoires), mais avec des possibilités surmultipliées. A partir de là, le lecteur accède naturellement et progressivement à la notion de processeur dont il peut approfondir le fonctionnement s'il fait l'effort d'étudier les exemples de programmation proposés.

→

La deuxième partie de l'ouvrage se consacre à la conception et à la réalisation d'une maquette susceptible de faciliter l'étude de la mise en œuvre et de la programmation des microprocesseurs. Cette maquette, réalisée sur un circuit imprimé simple face, est essentiellement constituée par le microprocesseur CDP 1802 (R.C.A.)

capable de fonctionner à une fréquence d'horloge égale à zéro (en pas-à-pas), couplé à une mémoire RAM. On s'est essentiellement attaché à élucider tous les problèmes d'ordre purement électronique qui peuvent se poser lors de la conception d'une telle maquette.

→ Dans la troisième partie de l'ouvrage, on analyse la procédure d'enregistrement et de déroulement d'un programme élémentaire. Cette procédure est l'objet d'une étude en pas-à-pas qui permet de comprendre, de façon claire et raisonnée, le principe de fonctionnement d'un microprocesseur. Ensuite on introduit progressivement les notions de branchement, de sous-programmes, d'interruption. L'assimilation de ce langage nouveau pour un électronicien est grandement facilitée par de nombreux exemples d'applications concrètes. Le lecteur constate alors de lui-même que le microprocesseur est un composant d'utilisation aisée dont la mise en œuvre ne nécessite aucune connaissance en informatique et ne requiert pas obligatoirement le support d'une structure logicielle encombrante.

→ Dans la quatrième partie enfin, on développe des applications et on s'attache à résoudre les problèmes d'interfaçage rencontrés lors de la conception d'un système automatique réalisant une fonction préalablement programmée. A l'issue de cette dernière étape, le lecteur peut envisager lui-même la conception d'un système bâti autour d'un microprocesseur et dispose en même temps de tous les outils propres à sa réalisation.

Nous espérons surtout que, grâce à cet ouvrage, bien des réticences seront vaincues, que le microprocesseur sortira enfin des régions ténébreuses dans lesquelles il a été tenu, permettant ainsi son libre accès à tout électronicien imaginatif et désireux d'agrandir prodigieusement le champ de ses investigations.

Première partie

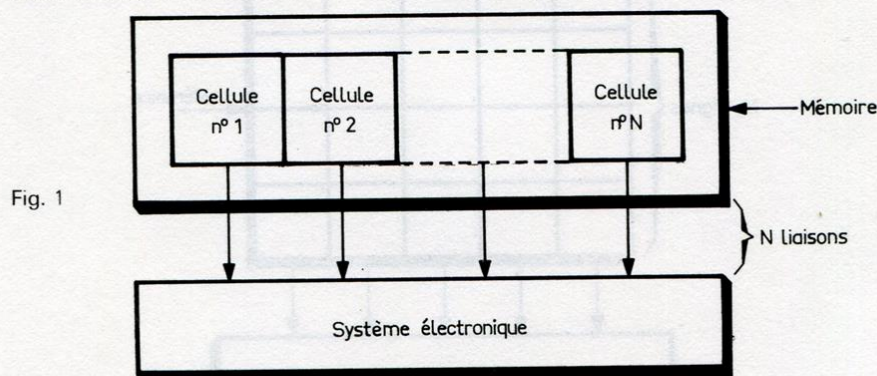
STRUCTURE ET FONCTIONNEMENT DES PROCESSEURS

Chapitre premier LES MÉMOIRES

I. Organisation d'une mémoire.

Une mémoire est un organe qui est capable de mémoriser une information qui lui a été fournie. Pour le type de mémoires qui nous intéresse, cette information est constituée par un ensemble de *niveaux logiques*. La plus petite mémoire que nous puissions imaginer, et que nous appellerons « cellule », est capable de mémoriser un seul niveau logique (soit 0, soit 1). La réunion de N cellules forme une mémoire de *capacité* N bits. ⁽¹⁾

Il faut tout de suite se rendre compte que les informations contenues dans une mémoire sont destinées à commander un système électronique ; par conséquent, il faut que ce système puisse *accéder* à chacun des bits mémorisés dans une cellule. Cet accès peut être matérialisé par un ensemble de liaisons électriques entre le système électronique, et chacune des sorties des cellules (fig. 1).



(1) *Bit* : unité d'information élémentaire. L'information élémentaire, susceptible d'être mémorisée dans une cellule, est matérialisée par un niveau logique égal soit à 0 soit à 1.

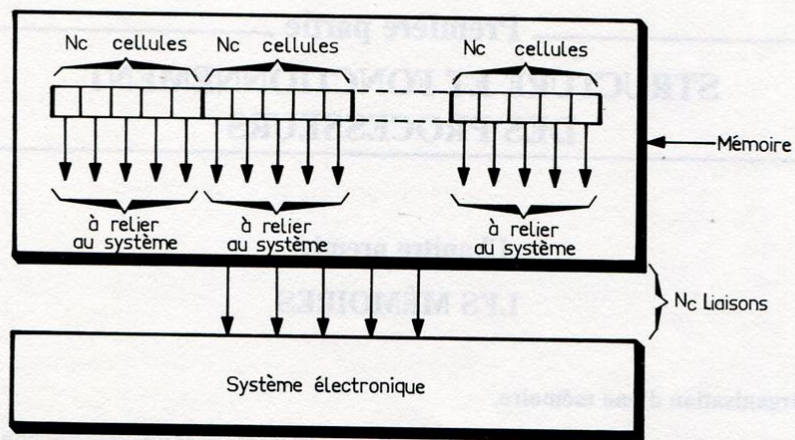


Fig. 2

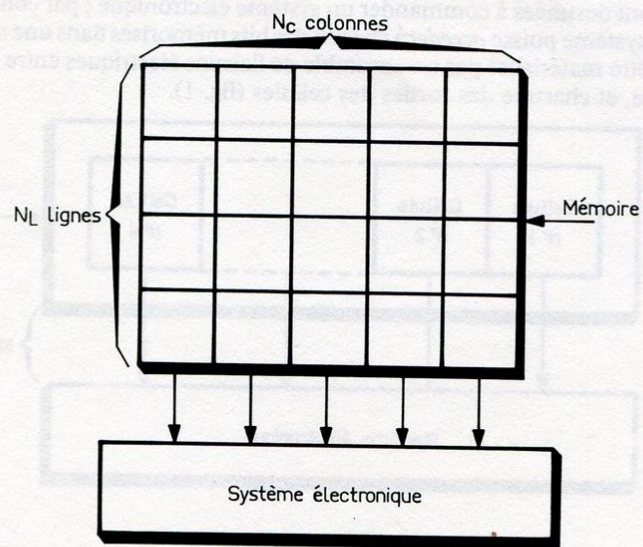


Fig. 3

Cependant cette solution est pratiquement inadmissible puisqu'elle nécessite autant de liaisons qu'il y a de cellules. En effet, voyez ce que cela donne si $N = 1024$! On peut lever la difficulté en acceptant de fournir au système électronique, non pas *toutes* les informations contenues dans la mémoire *en même temps*, mais seulement un certain nombre N_C typiquement égal à 1, 4 ou 8. Autrement dit si, par un procédé que nous détaillerons plus loin, on est capable de relier les sorties de tout ensemble de N_C cellules au système électronique, on simplifie considérablement la structure matérielle de l'ensemble mémoire-système, puisque le nombre de liaisons est limité à N_C (fig. 2).

Pour des raisons de commodité de langage, et aussi parce que telle est effectivement la structure physique du circuit intégré mémoire, tout groupe de N_C cellules est appelé *ligne*. Nous dirons donc qu'une mémoire est un ensemble de cellules organisées en N_L lignes et N_C colonnes (fig. 3). Cette organisation résulte de la nécessité de limiter à un nombre raisonnable les liaisons à un système électronique à commander, mais présente en contrepartie le défaut de n'autoriser qu'un accès séquentiel à la mémoire ; autrement dit, les *données* (les informations binaires) contenues dans toute ligne ne peuvent être fournies au monde extérieur que *successivement dans le temps*.

Une autre conséquence importante de cette organisation est l'obligation de disposer de signaux de commande de la mémoire, destinés à *sélectionner* la ligne dont les sorties sont à relier au système électronique à commander.

Une seule ligne étant sélectionnée à la fois, il est suffisant de disposer d'un nombre p de signaux de sélection tel que $2^p \geq N_L$. Par exemple, 8 signaux sont nécessaires et suffisants à la sélection d'une ligne parmi 256.

Chaque ligne est affectée d'un numéro qui est appelé son *adresse* ; l'ensemble des p fils de sélection de ligne qui supportent matériellement l'adresse transmise à la mémoire, constitue le *bus des adresses*.

II. L'accès à la mémoire.

Comment le *mot* binaire de p bits représentant l'adresse d'une ligne peut-il physiquement sélectionner une ligne parmi 2^p , c'est-à-dire par quel moyen l'information contenue dans la ligne adressée est-elle présentée sur les N_C liaisons au système électronique ?

Pour le comprendre, reportons-nous à la figure 4 qui représente une mémoire de capacité 8 bits organisée en $N_L = 4$ lignes et $N_C = 2$ colonnes. D'après ce que nous venons de dire, 2 signaux sont nécessaires et suffisants à la sélection d'une ligne parmi 4 ; ces 2 signaux sont supportés par les deux fils notés A_1 et A_0 , constituant le bus des adresses. A_1 et A_0 sont les entrées d'un décodeur à $2^2 = 4$ sorties notées L_1 , L_2 , L_3 et L_4 et, selon la combinaison présente en A_1 et A_0 l'une de ces 4 sorties est

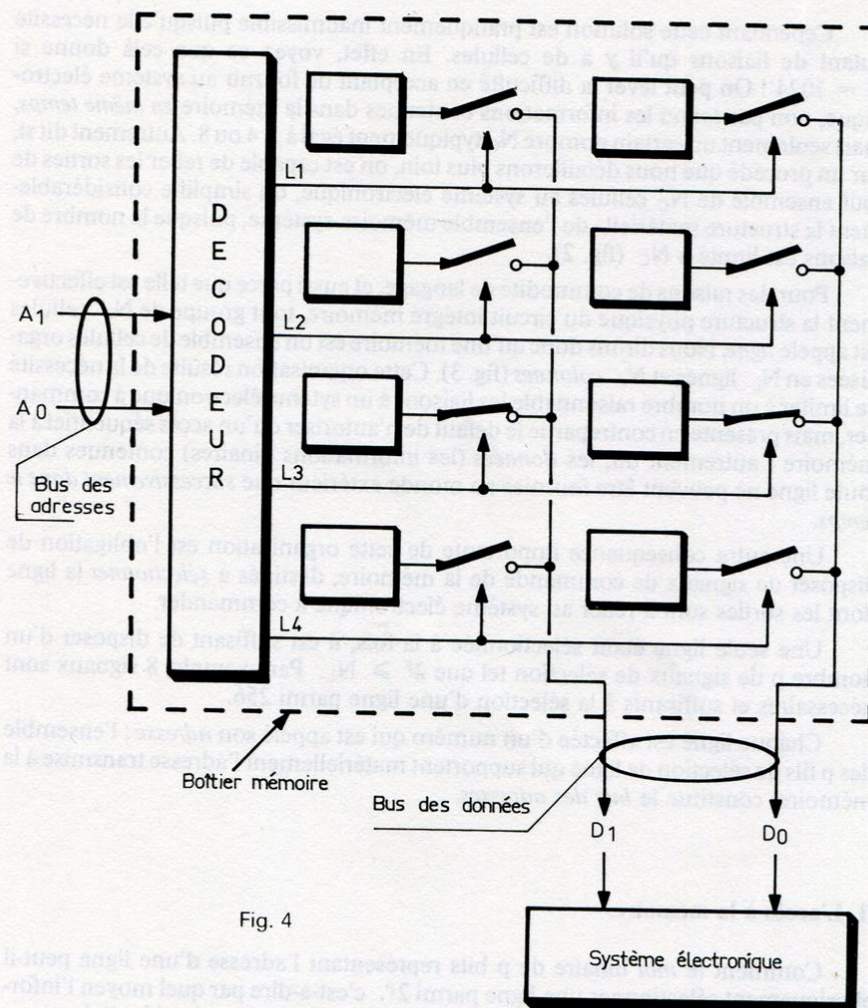


Fig. 4

imposée au niveau 0, les trois autres restant au niveau 1. La table de vérité du décodeur est donnée figure 5. Il est à noter que ce décodeur fait partie intégrante du boîtier mémoire, et lorsque par la suite nous parlerons de mémoire, il s'agira non seulement de la mémoire proprement dite c'est-à-dire des N cellules constitutives, mais du boîtier incluant le décodeur.

A ₁	A ₀	L ₁	L ₂	L ₃	L ₄
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Fig. 5

Le point essentiel à observer est que les sorties L₁, L₂, L₃, L₄ du décodeur sont destinées à commander des interrupteurs électroniques. Chaque interrupteur est constitué par un transistor que nous supposons passant (interrupteur fermé) si le signal L du décodeur est au niveau 0, et bloqué (interrupteur ouvert) si L est au niveau 1. Ces interrupteurs ont pour mission de « brancher » les sorties des cellules de la ligne adressée, et seulement celles-là, au système électronique à commander. On dit que les informations contenues dans la ligne concernée sont *lues* par le système ; l'ensemble des N_C fils de transmission sur lesquels circulent ces informations, notées D₁ et D₀ sur la figure 4, s'appelle *bus des données*.

Remarquez que les interrupteurs ont une double fonction :

- sélectionner une ligne et une seule parmi les N_L disponibles, dont le contenu est destiné à être lu par le système électronique.
- autoriser chacune des informations élémentaires contenue dans chacune des cellules constituant une colonne, à circuler sur la *même* voie de transmission. Ceci est rendu possible seulement parce qu'*une seule ligne est adressée à la fois*. En effet, si cela n'était pas le cas, les sorties de plusieurs cellules se trouveraient reliées entre elles, ce qui engendrerait des courts-circuits.

L'utilisation d'interrupteurs conduit à l'adoption d'un terme nouveau, celui de *logique 3-états* (Tri-State-Logic en anglais) ; qu'est-ce que cela veut dire ?

Supposez qu'un observateur soit placé sur l'un des fils du bus des données, par exemple le fil noté D₁, et chargé de déterminer quel est l'état logique de la sortie d'une cellule, par exemple celle correspondant à la première ligne. Si l'interrupteur est fermé, sa réponse sera qu'il y a 2 états logiques possibles, état 0 ou état 1. Mais si l'interrupteur est ouvert, sa réponse sera qu'il y a un troisième état pour lequel la sortie de la cellule est « débranchée » de son point d'observation ; nous dirons dans ce cas que la cellule est dans l'état « *haute impédance* » (H.I.).

L'utilisation de la logique 3-états est extrêmement avantageuse puisqu'elle simplifie considérablement les interconnexions entre les composants. Par exemple,

pour une mémoire de capacité 1 k bit (1 k bit = 1024 bits) organisée en 128 lignes et 8 colonnes. 7 fils d'adresse et 8 fils de données sont suffisants pour l'interconnexion avec un système électronique. En contrepartie, la lecture d'une donnée prendra du temps ; c'est-à-dire que dès l'instant où une adresse aura été placée sur le bus des adresses, il faudra attendre un certain temps appelé *temps d'accès*, avant que les sorties de la ligne adressée soient disponibles sur le bus des données.

Ce temps d'accès dépend de la technologie utilisée ; il est de quelques centaines de nanosecondes (ns) en technologie MOS et de quelques dizaines de ns en technologie TTL. Ce sont cependant les mémoires réalisées en technologie MOS qui sont essentiellement utilisées, car c'est elle qui permet la fabrication de composants à haute densité d'intégration.

III. Les mémoires RAM.

III - 1. Le signal d'écriture.

RAM veut dire « Random Access Memory », c'est-à-dire en traduction littérale « Mémoire à accès aléatoire ». Cette dénomination ne rend pas très bien compte du fait que ce type de mémoire peut être soumis à une succession d'écritures et de lectures, et ceci dans un ordre tout à fait indifférent.

La structure d'une cellule est celle du registre latch de capacité 1 bit de la figure 6 : DI est une donnée destinée à être écrite, et DO une donnée destinée à être lue :

– si $\overline{WR} = 0$, $DO = DI$; le registre est « transparent » à la donnée DI présentée en son entrée : la donnée DI est écrite dans la cellule.

– si $\overline{WR} = 1$, DO est insensible aux variations éventuelles de DI. Par ailleurs, l'état logique de DO est celui immédiatement précédant la transition $0 \rightarrow 1$ de \overline{WR} : la

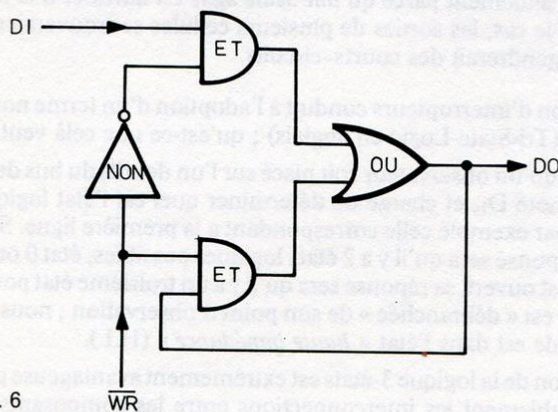


Fig. 6

donnée DI est mémorisée dans la cellule lors du front montant du signal \overline{WR} . Le signal \overline{WR} (Write) est le *signal d'écriture* et est actif au niveau bas ; c'est pourquoi il est surligné.

La figure 7 montre de quelle façon une donnée présentée sur le *bus d'entrée des données* DI, est écrite dans la ligne adressée et seulement dans celle-là. Pour plus de clarté, nous n'avons représenté qu'une mémoire de capacité 2 bits (2 lignes et 1 colonne). La voie de transmission sur laquelle circule le signal d'écriture \overline{WR} , est reliée aux entrées de commande $\overline{WR}1$ et $\overline{WR}2$ des cellules par l'intermédiaire d'interrupteurs commandés par les sorties $L1$ et $L2$ du décodeur. Par conséquent, seule la cellule ou plus généralement la ligne adressée, est écrite. On peut remarquer que la donnée DI est présente sur le *bus de sortie des données* DO lors de son écriture dans la mémoire.

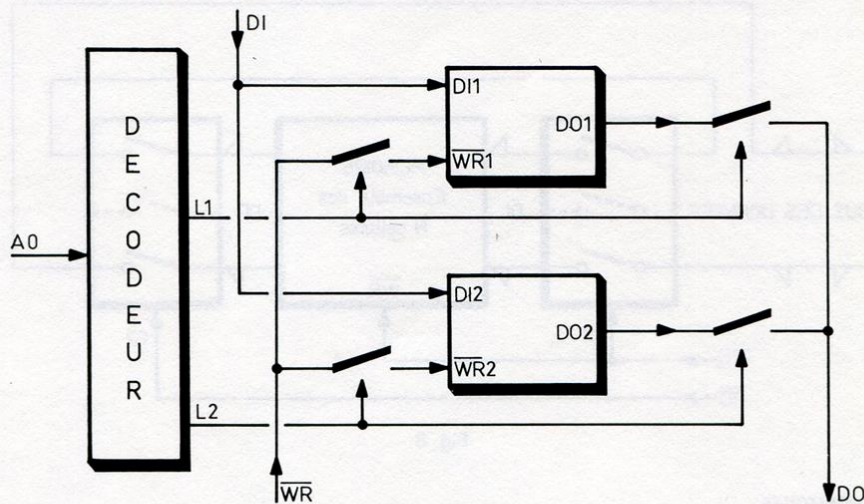


Fig. 7

III - 2. Les mémoires RAM à bus des données bidirectionnel.

Soit une mémoire RAM de capacité 1 k bit organisée en 128 lignes et 8 colonnes. L'adressage nécessite 7 fils, l'entrée des données 8 fils et la sortie des données 8 fils ; si l'on tient compte de l'alimentation et du signal d'écriture, nous sommes en présence d'un boîtier comportant 26 broches ce qui est prohibitif.

Pour ramener ce nombre à une valeur raisonnable, on est conduit à décider de véhiculer les données d'entrée et les données de sortie sur un *bus des données*

commun. Ce résultat est simplement obtenu en reliant DI à DO par l'intermédiaire d'interrupteurs commandés par C_1 et C_2 (fig. 8) :

- si $C_1 = 0$ et $C_2 = 1$, DI est égal à la donnée présentée sur le bus des données. Nous dirons que la mémoire est en position écriture. La donnée est effectivement écrite si $\overline{WR} = 0$.

- si $C_1 = 1$ et $C_2 = 0$, l'entrée DI de la mémoire est « débranchée » du bus des données, celui-ci supportant DO c'est-à-dire le contenu de la mémoire à la ligne adressée. Nous dirons que la mémoire est en position lecture.

Il apparaît que C_1 et C_2 sont des signaux d'écriture et de lecture. Nous pouvons donc relier \overline{WR} à C_1 et poser $C_2 = \overline{RD}$ (Read), signal d'écriture.

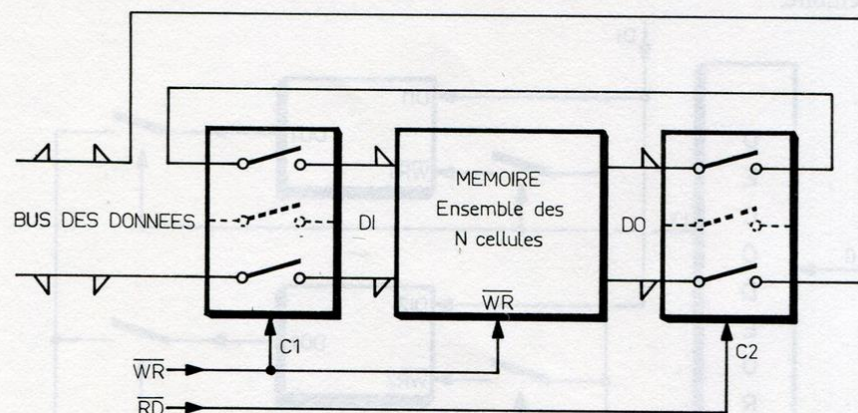


Fig. 8

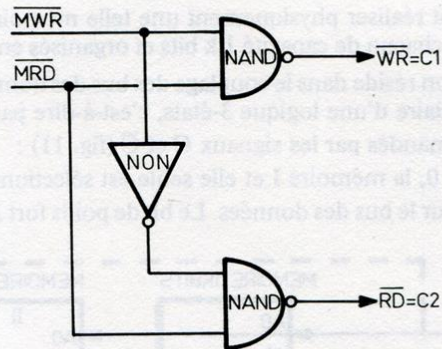
Remarques

1). Si $\overline{RD} = 1$ et $\overline{WR} = 1$, la mémoire est isolée du bus des données. On ne peut ni la lire ni l'écrire. Nous dirons qu'elle est dans l'état « haute impédance ».

2). Si $\overline{RD} = 0$ et $\overline{WR} = 0$, le bus des données supporte constamment DI et DO, ce qui interdit l'écriture de la mémoire. Cette combinaison doit donc être interdite, ce qui peut être réalisé par le circuit de la figure 9. \overline{MWR} (Memory Write) et \overline{MRD} (Memory Read) sont les signaux de contrôle du boîtier mémoire et sont tels que :

- si $\overline{MWR} = 0$ et $\overline{MRD} = 1$, la mémoire est en position écriture.
- si $\overline{MWR} = 1$ et $\overline{MRD} = 0$, la mémoire est en position lecture.
- si $\overline{MWR} = 1$ et $\overline{MRD} = 1$, ou si $\overline{MWR} = 0$ et $\overline{MRD} = 0$, la mémoire est en haute impédance.

Fig. 9



IV. La sélection de boîtier

IV. 1. Principe

Soit une mémoire de capacité 2 k bits organisée en 256 lignes et 8 colonnes (fig. 10). Le bus des adresses est constitué par 8 fils notés A_0 à A_7 ; A_0 est le bit de poids faible de l'adresse, et A_7 le bit de poids fort. Le bus des données constitué par 8 fils, supporte l'information à lire ou à écrire. La lecture et l'écriture sont contrôlés par les signaux \overline{MWR} et \overline{MRD} .

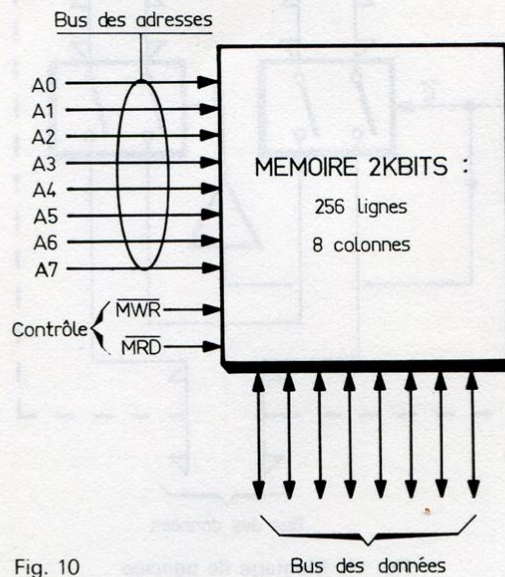


Fig. 10

Comment réaliser physiquement une telle mémoire si on ne dispose que de deux boîtiers, chacun de capacité 1 k bits et organisés en 128 lignes et 8 colonnes ?

La solution réside dans le couplage des bus des données de chacun des boîtiers par l'intermédiaire d'une logique 3-états, c'est-à-dire par l'intermédiaire des interrupteurs commandés par les signaux C et \bar{C} (fig. 11) :

– si $\bar{C} = 0$, la mémoire I et elle seule est sélectionnée, et par conséquent est « branchée » sur le bus des données. Le bit de poids fort A_7 de l'adresse étant relié à

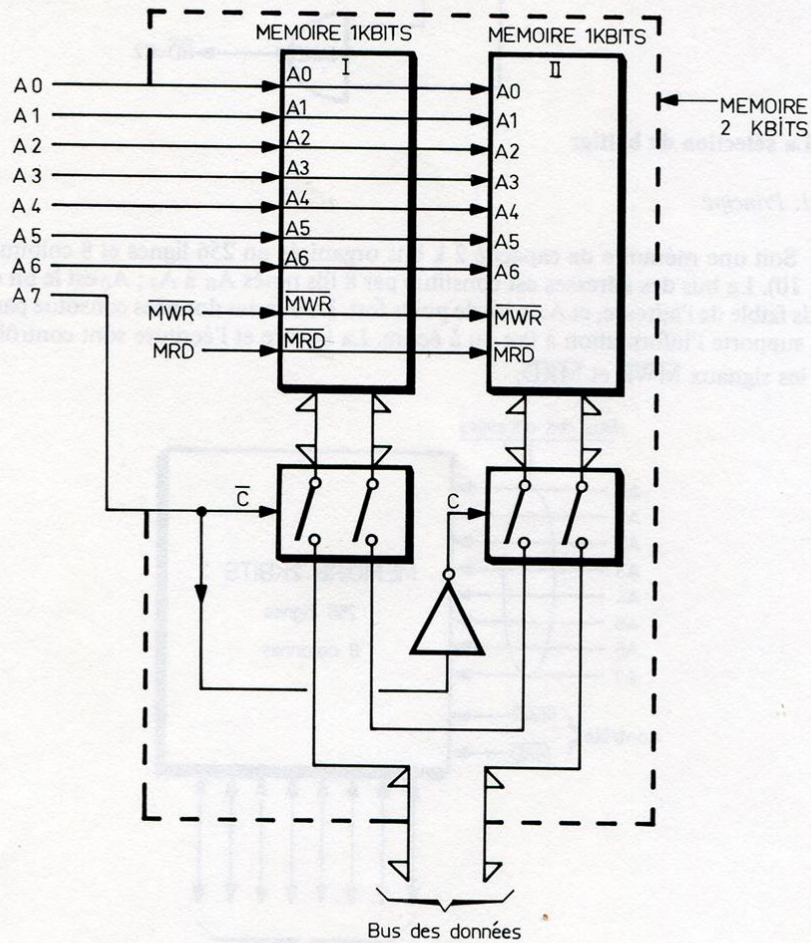


Fig. 11. Montage de principe

\overline{C} , la mémoire I est sélectionnée seulement quand l'adresse ($A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$) varie de (00000000) à (01111111). En notation hexa-décimale, nous dirons que la mémoire I réside aux adresses de 00 à 7F. (1)

– si $\overline{C} = 1$, c'est-à-dire quand l'adresse varie de (10000000) à (11111111), la mémoire II et elle seule est « branchée » sur le bus des données. La mémoire II réside aux adresses de 80 à FF.

Il apparaît donc que les signaux de commande \overline{C} et C sont des signaux de sélection de boîtier de mémoire. Ils sont habituellement notés \overline{CS} (ou CS), ce qui veut dire « Chip Select », c'est-à-dire sélection de boîtier.

IV - 2. Montage réel

En réalité, si l'on désire réaliser une mémoire de capacité 2 k bits avec deux boîtiers de capacité 1 k bits chacun, il n'est pas nécessaire de rajouter les interrupteurs de la figure 11. En effet, fixons notre attention sur la mémoire I par exemple :

– si $\overline{C} = 1$, la mémoire I doit être placée dans l'état haute impédance, ce qui peut être aisément effectué en s'arrangeant pour imposer sur les entrées de commande \overline{WR} et \overline{RD} de la figure 8 un niveau 1 lorsque $\overline{C} = 1$.

– si $\overline{C} = 0$, la mémoire I doit être sélectionnée, ce qui veut dire que les entrées de commande \overline{WR} et \overline{RD} doivent être à un niveau dépendant seulement de l'état logique des signaux de lecture et d'écriture $M\overline{RD}$ et $M\overline{WR}$.

La figure 12 montre comment il faut transformer le circuit de contrôle de la figure 9 pour répondre aux spécifications que nous venons d'énoncer. Ce circuit de

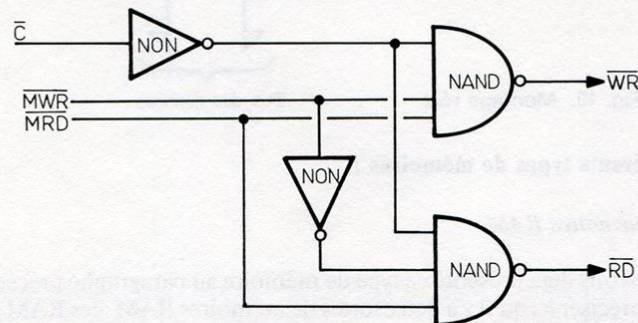


Fig. 12

(1) La conversion binaire-hexadécimal est indiquée dans le tableau 1 du chapitre 6.

contrôle fait habituellement partie intégrante du boîtier mémoire, et son principe peut être étendu à un nombre quelconque d'entrées « Chip Select ». En particulier, si l'on dispose de deux boîtiers mémoire de capacité 1 k bits présentant chacun une entrée \overline{CS}_1 et une entrée CS_2 , la figure 13 montre que la réalisation d'une mémoire de capacité 2 k bits ne nécessite aucun composant additionnel.

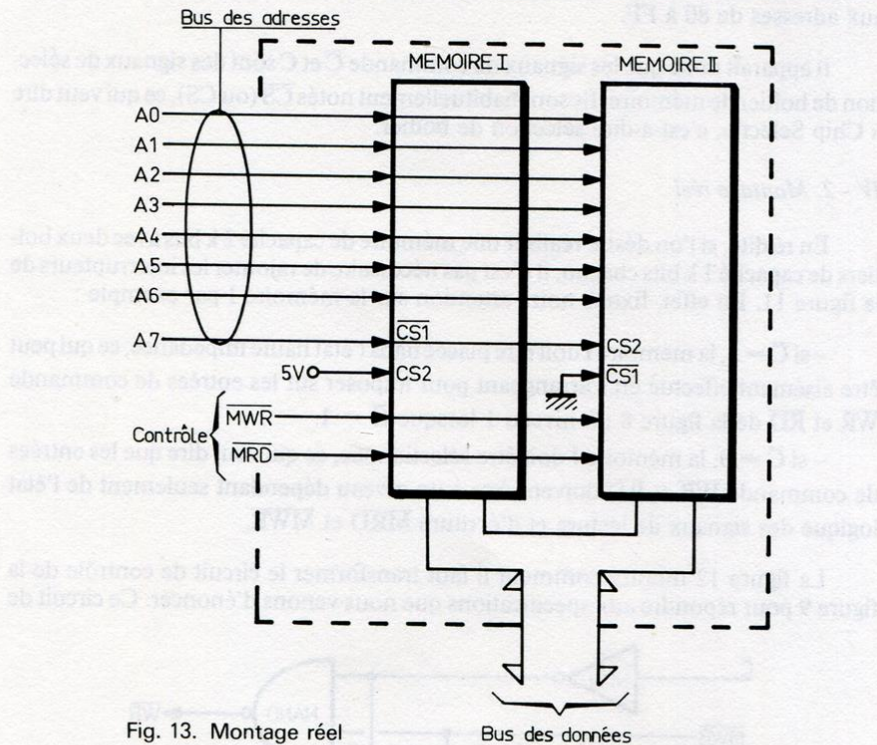


Fig. 13. Montage réel

V. Les différents types de mémoires :

V - 1. Les mémoires RAM

Nous avons déjà présenté ce type de mémoire au paragraphe précédent. Il faut cependant préciser ici qu'il y a deux sortes de mémoires RAM : les RAM statiques et les RAM dynamiques.

Les RAM statiques sont constituées par des cellules dont la structure est celle du registre latch de la figure 6 ; la mémoire conserve son contenu tant qu'elle est alimentée.

Les RAM dynamiques utilisent un principe de stockage de l'information tout à fait différent. En effet, c'est la capacité existant entre la Grille et le Drain d'un transistor MOS qui a pour fonction de mémoriser un niveau logique. La structure d'une cellule est donc extrêmement simple, ce qui permet d'obtenir une capacité de stockage relativement grande pour un seul boîtier. Typiquement, pour une surface donnée de la « puce » de silicium, une RAM dynamique présente une capacité de 16 k bits alors qu'une RAM statique présente seulement une capacité de 1 k bits. En contrepartie, une RAM dynamique conserve son contenu seulement si la perte de charge naturelle de la capacité du transistor MOS est compensée par sa recharge périodique. On dit qu'il est nécessaire de *rafraîchir* la mémoire ; ceci nécessite un complément de circuits dont la complexité et le coût se justifient à partir d'une taille mémoire de l'ordre de 32 k bits. En dessous de ce chiffre, il est plus simple et moins coûteux d'associer plusieurs boîtiers de RAM statiques de capacité moindre.

V - 2. Les mémoires ROM

ROM veut dire « Read Only Memory », autrement dit « Mémoire à Lecture Seule ». Plus précisément, il s'agit de mémoires que le constructeur a programmées une fois pour toutes, et que l'utilisateur ne peut que lire.

La structure d'une mémoire ROM est celle d'une matrice à diodes. La figure 14 représente une telle mémoire de capacité 16 bits, organisée en 4 lignes et 4 colonnes. Pour une adresse donnée présentée en entrée du décodeur, la ligne L correspondante est imposée au niveau 0 tandis que les autres restent au niveau 1. Par consé-

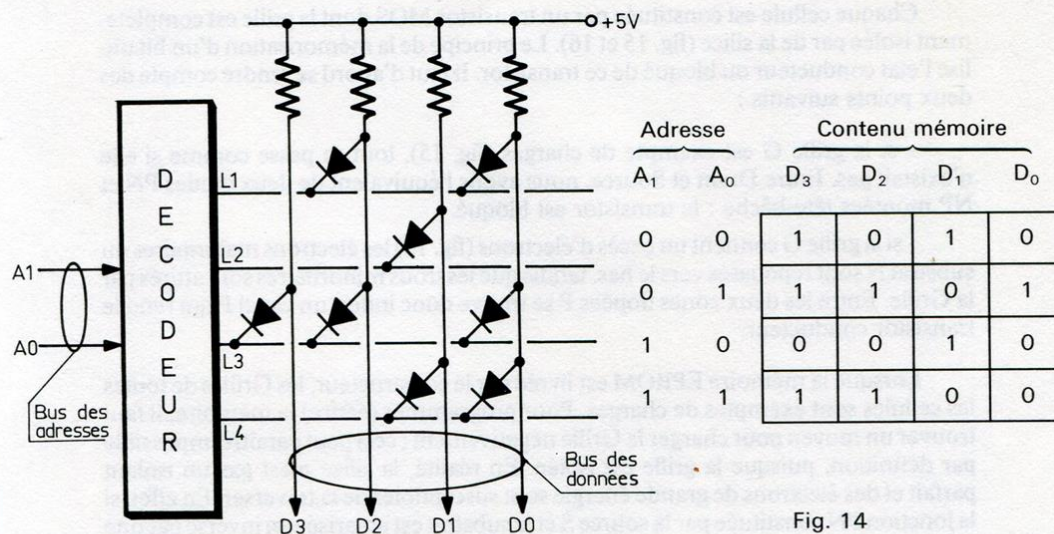


Fig. 14

quent, les diodes dont les cathodes sont reliées à la ligne sélectionnée sont conductrices tandis que toutes les autres restent bloquées. Ceci entraîne que les colonnes reliées aux anodes des diodes conductrices supportent un niveau logique 0 tandis que les autres supportent un niveau logique 1.

Compte tenu de la simplicité de structure des mémoires ROM, leur capacité est relativement grande pour un seul boîtier. (typiquement 8 à 16 k bits)

V - 3. Les mémoires PROM

PROM veut dire « Programmable ROM » ; il s'agit donc d'une ROM que l'utilisateur peut programmer.

La structure d'une PROM est celle d'une ROM, avec la différence que, en série avec les diodes sont insérés des fusibles tous intacts lors de la livraison du produit. La mémoire contient donc initialement des niveaux logiques 0.

Si, par un procédé que nous ne détaillerons pas, on est capable de vaporiser sélectivement ces fusibles, donc de les transformer en des circuits ouverts, on écrit des « 1 » dans la mémoire. Il est évident qu'il n'est pas possible de rectifier des erreurs de programmation.

V - 4. Les mémoires EPROM

EPROM veut dire « Erasable PROM » ; il s'agit donc d'une PROM que l'utilisateur peut effacer.

Chaque cellule est constituée par un transistor MOS dont la grille est complètement isolée par de la silice (fig. 15 et 16). Le principe de la mémorisation d'un bit utilise l'état conducteur ou bloqué de ce transistor. Il faut d'abord se rendre compte des deux points suivants :

- si la grille G est exempte de charges (fig. 15), tout se passe comme si elle n'existait pas. Entre Drain et Source, nous avons l'équivalent de deux diodes PN et NP montées tête-bêche : le transistor est bloqué.

- si la grille G contient un excès d'électrons (fig. 16) les électrons majoritaires du substrat N sont repoussés vers le bas, tandis que les trous minoritaires sont attirés par la Grille. Entre les deux zones dopées P se trouve donc induit un canal P qui rend le transistor conducteur.

Lorsque la mémoire EPROM est livrée par le constructeur, les Grilles de toutes les cellules sont exemptes de charges. Pour programmer (écrire) la mémoire, il faut trouver un moyen pour charger la Grille négativement ; ceci peut paraître impossible par définition, puisque la grille est isolée. En réalité, la silice n'est pas un isolant parfait et des électrons de grande énergie sont susceptibles de la traverser. En effet, si la jonction PN constituée par la source S et le substrat est polarisée en inverse par une

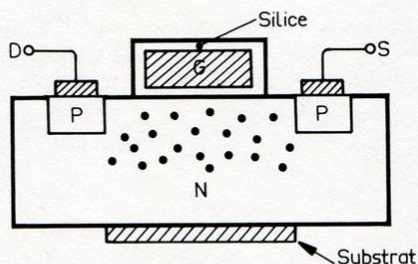


Fig. 15. Grille exempte de charges

Fig. 16. Grille chargée négativement

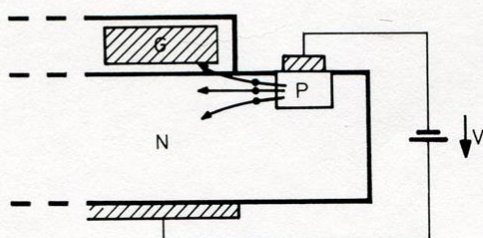
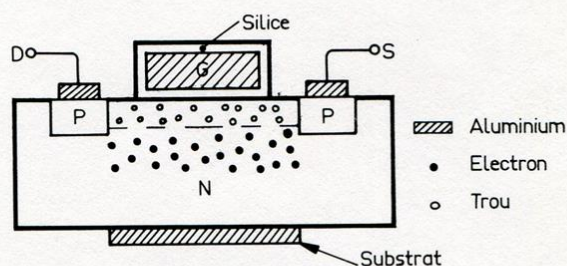


Fig. 17. Chargement de la grille

tension V de valeur importante (typiquement de 30 à 40 V), la zone P (fig. 17) diffuse dans le substrat N des électrons dont l'énergie est suffisante pour que quelques-uns d'entre eux atteignent la Grille. Lorsque la tension inverse V est supprimée, la Grille reste isolée : le bit correspondant est programmé.

Une mémoire EPROM programmée peut être effacée en vue de subir une nouvelle programmation. Pour effacer une EPROM, il faut décharger la Grille des transistors MOS. Ceci est réalisé de la façon suivante.

Une fenêtre ouverte à la puce de silicium constituant la mémoire, rend accessible toutes les grilles des cellules à un rayonnement extérieur. Si ce rayonnement est suffisamment énergétique, c'est-à-dire s'il est constitué par des rayons U.V. en quantité adéquate, son effet est de transférer l'excès de charge supporté par la grille dans le substrat : la mémoire est effacée et, remarquez-le, complètement, ce qui veut dire qu'il n'est pas possible de l'effacer sélectivement.

Chapitre 2

AUTOMATE PROGRAMMABLE

Le cœur des systèmes microprogrammés est constitué par des *automates*, systèmes logiques dont le rôle est d'accomplir (ou *exécuter*) les différentes fonctions pour lesquelles ils ont été programmés.

Le présent chapitre et le suivant ont pour objet de mettre en évidence le principe de tels organes et de faire comprendre le mécanisme permettant l'exécution d'une série d'ordres (ou *instructions*)

I. Description et fonctionnement d'un automate programmable.

Considérons le système représenté sur la figure 1. Son fonctionnement est le suivant :

I - 1. A une époque déterminée t , le compteur est chargé par le nombre binaire n . Ce nombre n est le rang d'une ligne de la mémoire dans laquelle est inscrit le mot

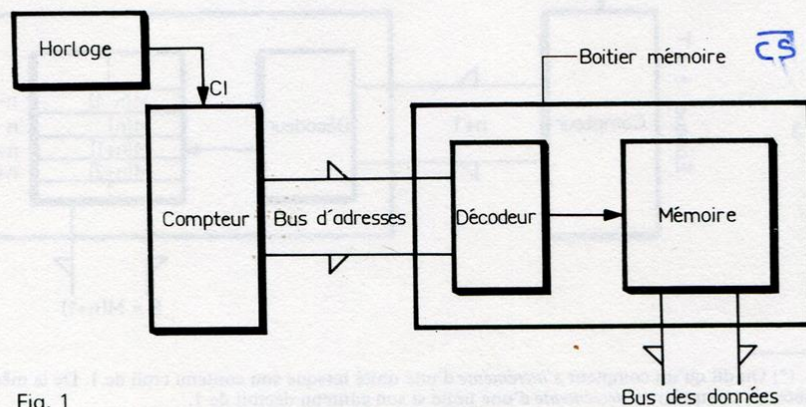
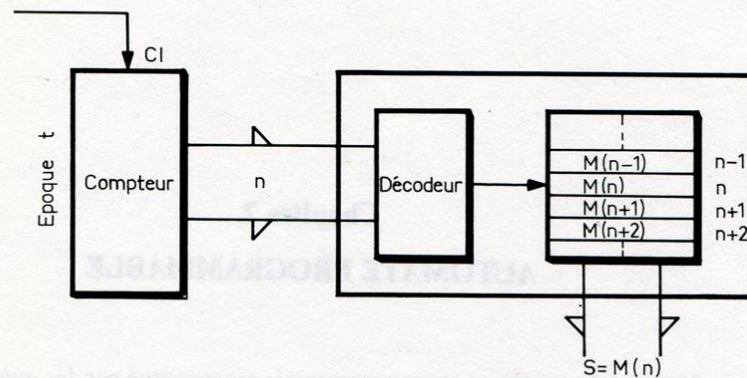


Fig. 1

Fig. 2



$M(n)$. Si cette mémoire est en fonction « lecture » (read), le mot $M(n)$ apparaît sur les lignes de sortie de la mémoire dont l'ensemble est appelé *bus des données* (data bus)

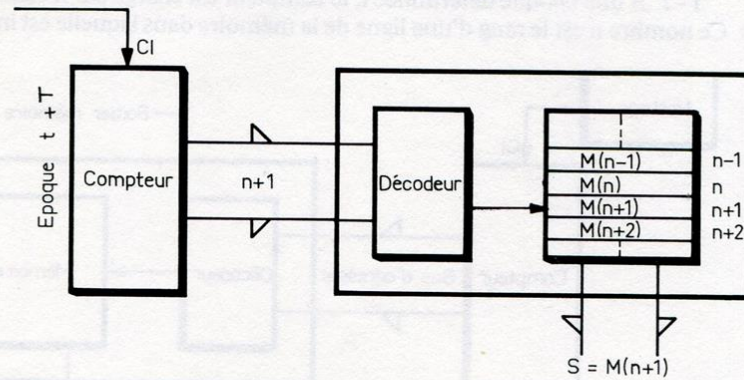
Cette situation est décrite par le schéma de la figure 2. On dit que le compteur *pointe* la ligne d'adresse n de la mémoire ou que le compteur *adresse* la mémoire à la ligne n .

I - 2. Quand apparaît en Cl le front montant d'une impulsion délivrée par l'horloge, le compteur s'incrémente (*) d'une unité et vient donc pointer la ligne mémoire d'adresse $n+1$.

Sur le bus des données apparaît alors le mot $M(n+1)$ inscrit dans la mémoire à la ligne d'adresse $n+1$.

Cette situation est décrite par le schéma de la figure 3.

Fig. 3



(*) On dit qu'un compteur *s'incrémente* d'une unité lorsque son contenu croît de 1. De la même manière, un compteur se *décrémente* d'une unité si son contenu décroît de 1.

I - 3. Les impulsions d'horloge suivantes incrémentent à nouveau le compteur qui pointe successivement les lignes mémoire d'adresses $n + 2$, $n + 3$, $n + 4$, ... et les mots $M(n + 2)$, $M(n + 3)$, $M(n + 4)$, ... apparaissent successivement sur le bus des données.

I - 4. *En résumé*, ce système permet d'obtenir successivement sur le bus des données, au rythme de l'horloge, les mots binaires contenus dans la mémoire aux adresses n , $n + 1$, $n + 2$, ... Chacun de ces mots constitue ce que l'on appelle une *microinstruction*. Chaque mot est formé d'un certain nombre de bits qui sont autant de *microactions* utilisées pour la commande de dispositifs logiques.

Le système étudié est donc capable de fournir automatiquement sur le bus des données une séquence de microactions qui aura été préalablement enregistrée (*programmée*) dans la mémoire.

II. Exemples d'utilisation

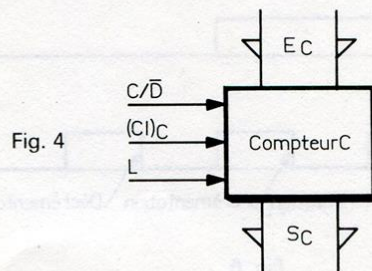
II - 1. Commande d'un compteur

Supposons que l'on veuille réaliser sur un compteur C, à partir d'un instant t où le compteur d'adressage pointe l'adresse n de la mémoire, la séquence suivante, dans l'ordre indiqué :

- chargement
- incrémentation
- incrémentation
- décrémentation
- décrémentation

A cet effet, le compteur C utilisé possède trois entrées de commande notées L , $(Cl)_C$, C/\bar{D} (fig. 4).

Si $L = 1$, $S_C = E_C$: le compteur ne remplit pas son office de comptage. Il est « transparent ».



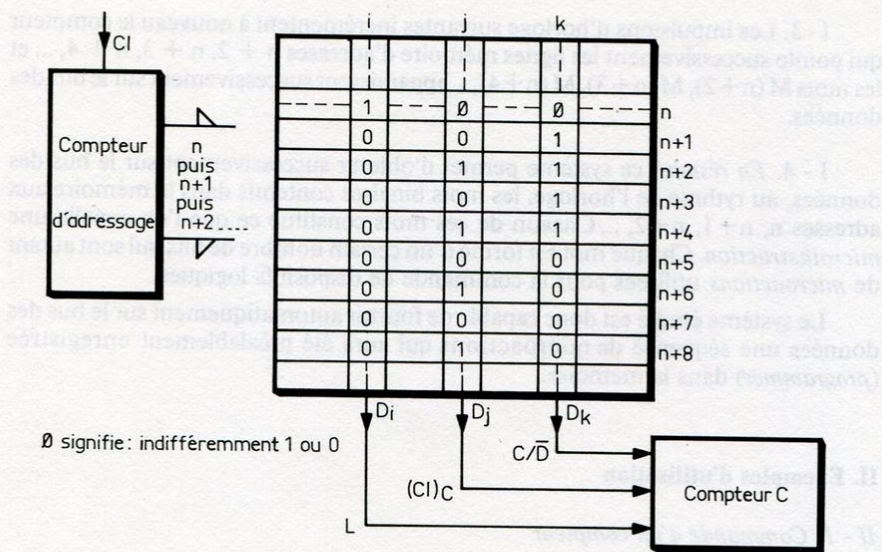


Fig. 5

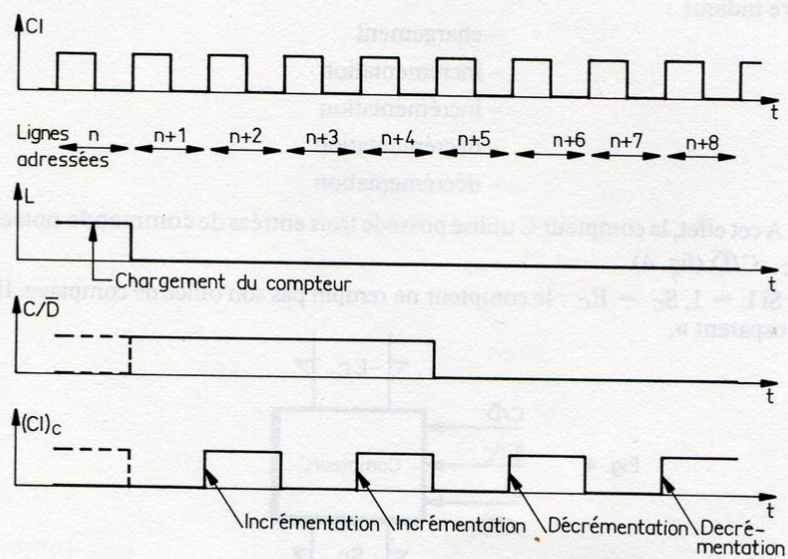


Fig. 6

Si $L = 0$, le compteur remplit son office de comptage ou de décomptage, à partir du nombre binaire mémorisé lors de la transition 1 à 0 de L , en s'incrémentant d'une unité à chaque front montant de $(Cl)_C$ si $C/\overline{D} = 1$ ou en se décrémentant d'une unité à chaque front montant de $(Cl)_C$ si $C/\overline{D} = 0$.

Trois microactions sont donc nécessaires pour réaliser la séquence désirée. Les éléments D_i, D_j, D_k du bus des données de la mémoire seront reliés respectivement aux entrées de commande $L, (Cl)_C$ et C/\overline{D} du compteur C . Les autres éléments du bus des données peuvent être utilisés pour d'autres commandes simultanées que l'on n'envisage pas ici. Le contenu des colonnes i, j, k à partir de l'adresse n sera celui de la figure 5. Le diagramme des temps correspondant est fourni sur la figure 6.

II - 2. Commande d'un système permettant d'effectuer $x + ay$ (a entier naturel)

On désire programmer la mémoire d'un automate programmable destiné à fournir les signaux de commande du système de la figure 7 comportant un additionneur, trois registres R, T et D et un multiplexeur. En plus des entrées de commande

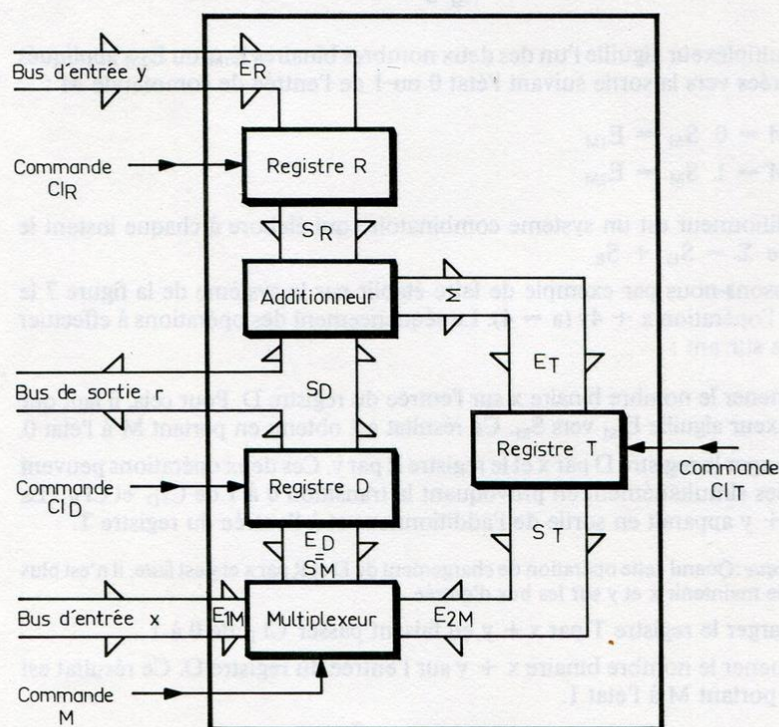


Fig. 7

Cl_R , Cl_T , Cl_D et M le système comporte deux bus d'entrée des données x et y à traiter et un bus de sortie du résultat r recherché.

Le registre X ($X = R$ ou T ou D) charge des données E_X présentes sur les entrées de chargement parallèle au moment de la transition 0 à 1 du signal de commande appliqué sur l'entrée Cl_X . La transition 1 à 0 de Cl_X n'a pas d'effet sur la sortie S_X (fig. 8).

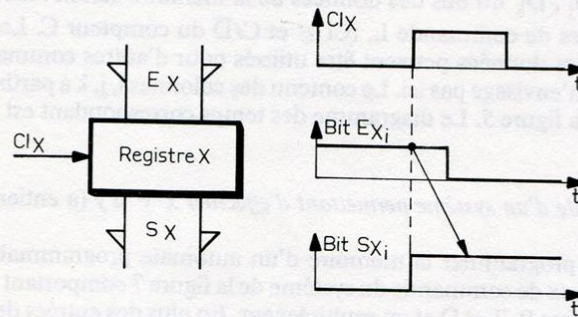


Fig. 8

Le multiplexeur aiguille l'un des deux nombres binaires E_{1M} ou E_{2M} appliqués sur les entrées vers la sortie suivant l'état 0 ou 1 de l'entrée de commande M :

- si $M = 0$ $S_M = E_{1M}$
- si $M = 1$ $S_M = E_{2M}$

L'additionneur est un système combinatoire qui élabore à chaque instant le mot binaire $\Sigma = S_D + S_R$

Proposons-nous par exemple de faire établir par le système de la figure 7 le résultat de l'opération $x + 4y$ ($a = 4$). Le séquencement des opérations à effectuer peut être le suivant :

- a) Amener le nombre binaire x sur l'entrée du registre D . Pour cela, il faut que le multiplexeur aiguille E_{1M} vers S_M . Ce résultat est obtenu en portant M à l'état 0.
- b) Charger le registre D par x et le registre R par y . Ces deux opérations peuvent être réalisées simultanément en provoquant la transition 0 à 1 de Cl_D et Cl_R . La somme $x + y$ apparaît en sortie de l'additionneur et à l'entrée du registre T .

Remarque : Quand cette opération de chargement de D et R par x et y est faite, il n'est plus nécessaire de maintenir x et y sur les bus d'entrée.

- c) Charger le registre T par $x + y$ en faisant passer Cl_T de 0 à 1
- d) Amener le nombre binaire $x + y$ sur l'entrée du registre D . Ce résultat est obtenu en portant M à l'état 1.

Remarque : Pour gagner une ligne de programme, on peut porter M à 1 plus tôt, par exemple dès que le registre D a été chargé par x.

Pour la suite du fonctionnement, on conservera $M = 1$ de manière que $E_D = S_M = E_{2M} = S_T$

e) Charger le registre D par $x + y$ présent en E_A en faisant passer Cl_D à 1.

Remarque : Le résultat partiel $x + y$ est alors présent sur le bus de sortie du système.

La somme $\Sigma = S_D + S_R = (x + y) + y = x + 2y$ apparaît alors en sortie de l'additionneur et sur l'entrée du registre T.

f) Charger le registre T par $x + 2y$. Comme on a conservé $M = 1$ on obtiendra $S_M = E_D = S_T = x + 2y$

g) Charger le registre D par $x + 2y$. Le résultat partiel $x + 2y$ est alors présent sur le bus de sortie du système et $\Sigma = (x + 2y) + y = x + 3y$

h) Charger le registre T par $x + 3y$

i) Charger le registre D par $x + 3y$. Le résultat partiel $x + 3y$ est alors présent sur le bus de sortie du système et $\Sigma = (x + 3y) + y = x + 4y$ ce qui est la quantité recherchée.

n	---	0	---	0	---	0	---	0
n + 1		1		1		0		0
n + 2		0		0		1		1
n + 3		1		0		0		1
n + 4		0		0		1		1
n + 5		1		0		0		1
n + 6		0		0		1		1
n + 7		1		0		0		1
n + 8		0		0		1		1
n + 9		1		0		0		1

↓
 $D_i = Cl_D$

↓
 $D_j = Cl_R$

↓
 $D_k = Cl_T$

↓
 $D_l = M$

Commentaire

$E_D = S_M = x$

$S_D = x \quad S_R = y \quad \Sigma = x + y$

$E_D = S_T = x + y$

$S_D = x + y \quad \Sigma = x + 2y$

$E_D = S_T = x + 2y$

$S_D = x + 2y \quad \Sigma = x + 3y$

$E_D = S_T = x + 3y$

$S_D = x + 3y \quad \Sigma = x + 4y$

$E_D = S_T = x + 4y$

$S_D = x + 4y \quad \Sigma = x + 5y$

Fig. 9

Pour obtenir ce résultat sur le bus de sortie du système il faut encore :

j) Charger le registre T par $x + 4y$

k) Charger le registre D par $x + 4y$.

Pour exécuter cette séquence, il faut agir sur les quatre commandes Cl_D , Cl_R , Cl_T et M. On utilisera à cet effet quatre colonnes de la mémoire correspondant aux sorties D_i , D_j , D_k et D_l .

Si l'on suppose que le programme de cette séquence débute à la ligne d'adresse n de la mémoire, le contenu de cette mémoire à partir de l'adresse n et dans les colonnes i, j, k et l sera celui représenté sur la figure 9. La figure 10 représente le chronogramme de fonctionnement du système.

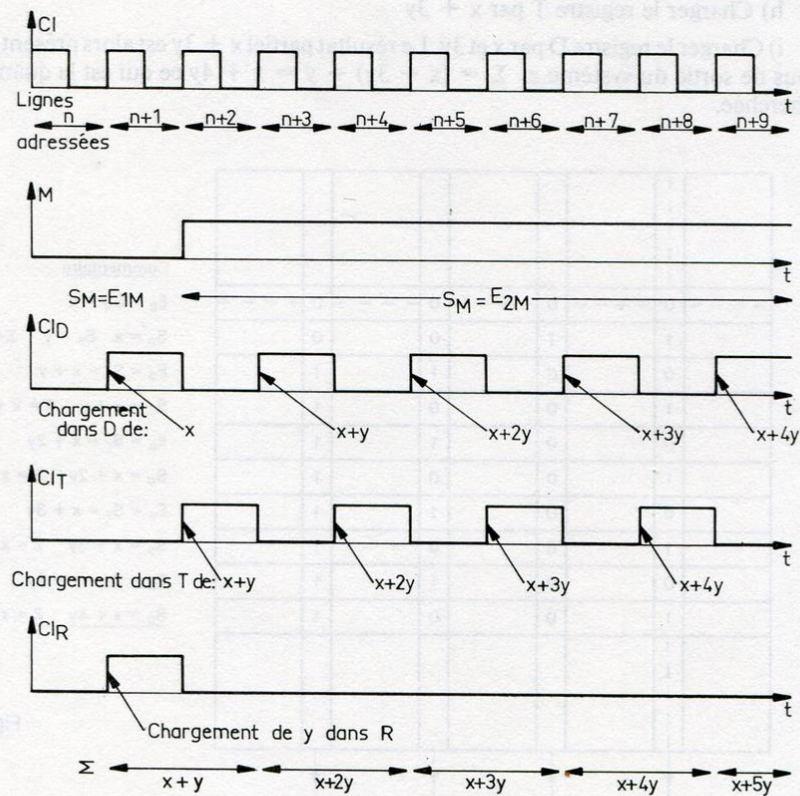


Fig. 10

On aura noté que les trois registres ne jouent pas des rôles équivalents :

- le registre R sert à recevoir et conserver la donnée y
- le registre D sert d'une part à recevoir la donnée x (comme le registre R) mais aussi à recevoir les résultats partiels $x + y$, $x + 2y$, $x + 3y$ qu'il fournit au bus de sortie. Un tel registre est appelé *accumulateur*.
- le registre T sert de tampon entre l'additionneur et le registre D quand $M = 1$ et met à l'abri des aléas.

III. Assimilation

La description et le développement des exemples précédents ont permis de comprendre le rôle et le fonctionnement d'un automate programmable. Dans le double but d'asseoir plus solidement les acquisitions qui viennent d'être faites et surtout de familiariser dès à présent le lecteur avec des structures intervenant dans le microprocesseur COSMAC étudié dans cet ouvrage, on se propose de présenter quelques autres exemples de programmation d'un automate. Les systèmes à commander, d'abord simples, seront petit à petit associés pour former des systèmes plus complexes.

III - 1. Exemples de programmation de fonctions élémentaires

a) chargement du contenu d'un registre R dans un registre D

Symboliquement, nous noterons cette opération : $R \rightarrow D$ (le contenu de R est chargé dans D)

On dispose de deux registres R et D dont l'entrée et la sortie sont reliés, par l'intermédiaire d'une logique trois états dont le principe a été étudié dans le chapitre 1, à un même bus, le *bus des données* (fig. 11) Lorsque $(\overline{RD})_X = 0$ (avec $X = R$ ou D) et à condition que $(\overline{WR})_X = 1$, la sortie du registre X est reliée au bus des données. Lorsque $(\overline{WR})_X = 0$ et à condition que $(\overline{RD})_X = 1$, c'est l'entrée du registre X qui est reliée au bus des données. Par conséquent, la combinaison $(\overline{RD})_X = 0$ et

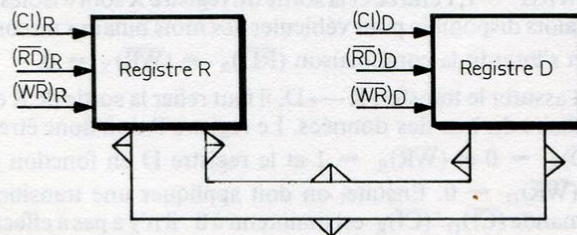
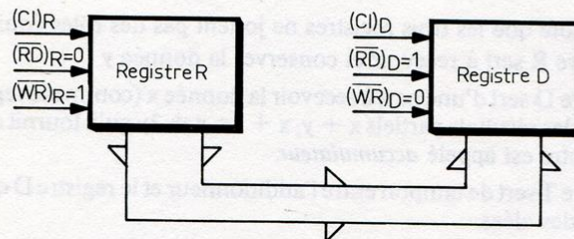


Fig. 11

Bus des données bidirectionnel



Le sens de transfert des données sur le bus est de R vers D

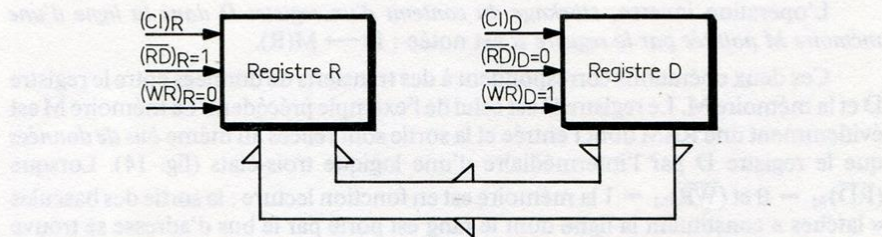
	$(\overline{RD})_R$	$(\overline{WR})_R$	$(CI)_R$	$(\overline{RD})_D$	$(\overline{WR})_D$	$(CI)_D$	Commentaire :
	1						
R → D	0	1	0	1	0	0	Réalisation des liaisons avec le bus des données (détermination du sens du transfert)
	0	1	0	1	0	1	Chargement de D
	1						
	1						
	1						
	1						

R → D

Fig. 12

$(\overline{WR})_X = 1$ place le registre X en fonction écriture (write). De même lorsque $(\overline{RD})_X = 1$ et $(\overline{WR})_X = 0$ le registre X est en fonction lecture (read) : il suffit alors de provoquer une transition 0 à 1 de $(CI)_X$ pour que le mot binaire présent sur le bus des données soit chargé sur le registre X. Il est important de noter que lorsque $(\overline{RD})_X = 1$ et $(\overline{WR})_X = 1$, l'entrée et la sortie du registre X sont « isolés » du bus des données qui est alors disponible pour véhiculer des mots binaires ne concernant pas le registre X. On s'interdit la combinaison $(\overline{RD})_X = (\overline{WR})_X = 0$.

Si l'on veut assurer le transfert R → D, il faut relier la sortie de R et l'entrée de D par l'intermédiaire du bus des données. Le registre R doit donc être en fonction lecture soit $(\overline{RD})_R = 0$ et $(\overline{WR})_R = 1$ et le registre D en fonction écriture soit $(\overline{RD})_D = 1$ et $(\overline{WR})_D = 0$. Ensuite, on doit appliquer une transition 0 à 1 sur l'entrée de commande $(CI)_D$. $(CI)_R$ est maintenu à 0 : il n'y a pas à effectuer de chargement de R.



Le sens de transfert des données sur le bus est de D vers R

	$(\overline{RD})_R$	$(\overline{WR})_R$	$(CI)_R$	$(\overline{RD})_D$	$(\overline{WR})_D$	$(CI)_D$	Commentaire :
	1	0	0	0	1	0	
$D \rightarrow R$	1	0	1	0	1	0	Chargement de R
	1	0	0	0	1	0	
	1	0	0	0	1	0	

$D \rightarrow R$

Fig. 13

Le sens de transfert des données, le contenu de la mémoire de l'automate pour réaliser cette fonction apparaissent sur la figure 12.

Le lecteur comprendra aisément que l'opération inverse, c'est-à-dire le *chargement du contenu du registre D dans le registre R* (symboliquement noté $D \rightarrow R$) se fait de la même manière. Le sens de transfert des données, le contenu de la mémoire de l'automate apparaissent sur la figure 13.

b) *Chargement du contenu de la ligne d'une mémoire M pointée par un registre R dans un registre D*

Cette opération sera notée symboliquement : $M(R) \rightarrow D$, $M(R)$ signifiant : contenu de la mémoire M à la ligne adressée par le registre R.

Il importe de ne pas confondre cette mémoire M, partie du système à commander, avec la mémoire de l'automate destiné à assurer la commande.

L'opération inverse, *stockage du contenu d'un registre D dans la ligne d'une mémoire M pointée par le registre R* est notée : $D \rightarrow M(R)$.

Ces deux opérations correspondent à des transferts de données entre le registre D et la mémoire M. Le registre D est celui de l'exemple précédent. La mémoire M est évidemment une RAM dont l'entrée et la sortie sont reliées au même *bus de données* que le registre D par l'intermédiaire d'une logique trois-états (fig. 14). Lorsque $(\overline{RD})_M = 0$ et $(\overline{WR})_M = 1$ la mémoire est en fonction lecture : la sortie des bascules « latches » constituant la ligne dont le rang est porté par le bus d'adresse se trouve reliée au bus des données et le mot binaire présent sur cette ligne est également présent sur le bus des données. Lorsque $(\overline{RD})_M = 1$ et $(\overline{WR})__M = 0$, la mémoire est en fonction écriture : l'entrée des bascules de la ligne pointée est reliée au bus des données en même temps que ces bascules sont rendues « transparentes » de manière que le mot présent sur le bus des données soit transféré sur l'entrée des bascules et en

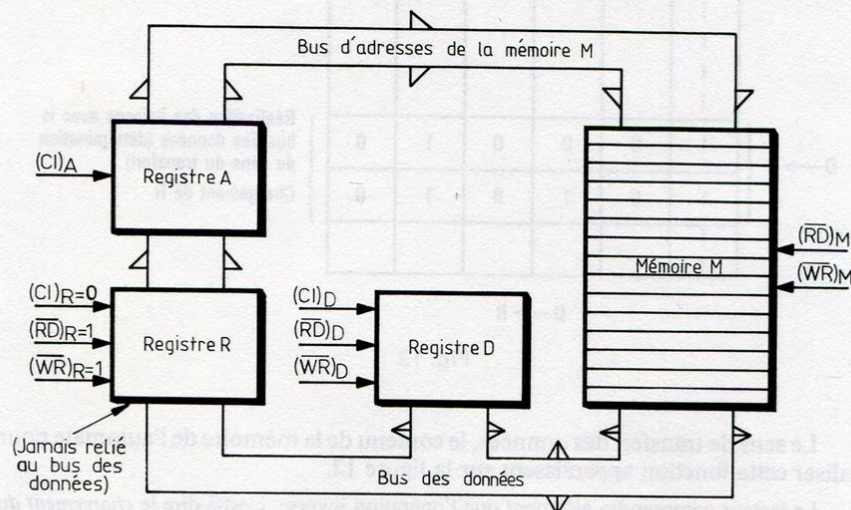


Fig. 14

même temps chargé dans ces bascules. La combinaison $(\overline{RD})_M = 1$ et $(\overline{WR})_M = 1$ (mémoire ni en lecture, ni en écriture) « isole » la RAM du bus des données. En outre, l'adressage de la mémoire M se fait par l'intermédiaire d'un registre A dit *registre d'adresse* qui charge la donnée présente à son entrée (contenu du registre R) au moment de la transition 0 à 1 de $(CI)_A$.

Au cours de ces deux opérations de transfert entre la mémoire M et le registre D, le registre R est « isolé » du bus des données en maintenant $(\overline{RD})_R$ et $(\overline{WR})_R$ à

l'état logique 1. On verra plus loin que sa sortie effective est néanmoins reliée à l'entrée d'un registre A. $(CI)_R$ reste à 0 : il n'y a pas à procéder à un chargement du registre R.

Pour assurer le transfert $M(R) \rightarrow D$, on peut commencer par adresser correctement la mémoire M en chargeant le registre A par le contenu du registre R ; il suffit pour cela de provoquer la transition 0 à 1 de $(CI)_A$. Au cours de cette opération, la mémoire M et le registre D peuvent être déconnectés du bus des données en imposant $(\overline{RD})_M = 1, (\overline{WR})_M = 1, (\overline{RD})_D = 1$ et $(\overline{WR})_D = 1$. Puis il faut relier la sortie de la mémoire M à l'entrée du registre D par l'intermédiaire du bus des données en fai-

	$(\overline{RD})_D$	$(\overline{WR})_D$	$(CI)_D$	$(\overline{RD})_M$	$(\overline{WR})_M$	$(CI)_A$	Commentaire :
	1						
	1	1	0	1	1	0	
$M(R) \rightarrow D$	1	1	0	1	1	1	Adressage de la mémoire par le contenu de R
	1	0	0	0	1	0	Réalisation des liaisons au bus des données (établissement du sens du transfert)
	1	0	1	0	1	0	Chargement de D
	1						

Fig. 15

$M(R) \rightarrow D$

	$(\overline{RD})_D$	$(\overline{WR})_D$	$(CI)_D$	$(\overline{RD})_M$	$(\overline{WR})_M$	$(CI)_A$	Commentaire :
	1						
	1	1	0	1	1	0	
$D \rightarrow M(R)$	1	1	0	1	1	1	Adressage de la mémoire par le contenu de R
	0	1	0	1	0	0	Réalisation des liaisons au bus des données et chargement de M
	1						
	1						

Fig. 16

$D \rightarrow M(R)$

sant $(\overline{RD})_M = 0$, $(\overline{WR})_D = 0$, $(\overline{WR})_M$ et $(\overline{RD})_D$ étant maintenus à 1. Le bus des données supporte alors $M(R)$ présent également à l'entrée effective du registre D ; $M(R)$ est chargé dans le registre D en faisant passer $(CI)_D$ de 0 à 1. La partie utile du contenu de la mémoire de l'automate est représenté sur la figure 15.

Le transfert $D \rightarrow M(R)$ se fait de manière identique : adressage préalable de la mémoire M en chargeant le registre A puis connexion de la sortie de D à l'entrée des bascules de la ligne mémoire pointée par l'intermédiaire du bus des données en faisant $(RD)_D = 0$ et $(\overline{WR})_M = 0$. La programmation de ces différentes phases dans la mémoire de l'automate apparaît sur la figure 16. Il n'y a pas à agir sur $(CI)_D$. $D \rightarrow M(R)$ nécessite une ligne de moins que $M(R) \rightarrow D$; ceci est dû à la différence de nature entre les bascules constituant le registre D et la mémoire M.

c) *Chargement du contenu de la ligne mémoire M pointée par un registre R dans ce registre R.* Notation : $M(R) \rightarrow R$.

Il s'agit d'une opération très proche de l'opération $M(R) \rightarrow D$. Le contenu préalable de R est chargé dans le registre A : la ligne pointée de la mémoire M est située à l'adresse correspondant au contenu du registre R. Le mot inscrit à cette adresse est alors chargé dans R. On comprendra par la suite l'utilité de cette opération.

Le registre d'adresse A dont l'utilité n'apparaissait pas dans l'exemple précédent est ici indispensable ; il sert de tampon entre le registre R et la mémoire M ; il conserve un adressage correct de la mémoire M pendant le chargement du registre R.

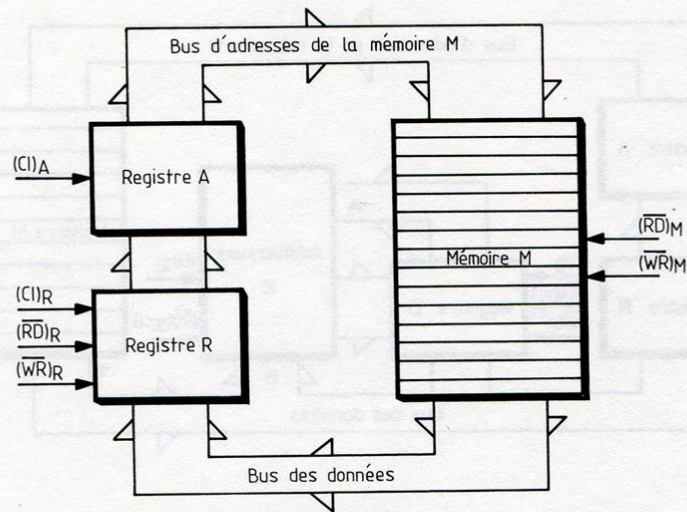
Au cours de cette opération, le registre D n'est pas utilisé. Il est déconnecté du bus des données en maintenant $(RD)_D = 1$ et $(\overline{WR})_D = 1$. La figure 17 représente la partie utile du système à commander et la partie utile du contenu de la mémoire de l'automate.

d) *Addition du contenu d'un registre D et du contenu de la ligne d'une mémoire M pointée par un registre R et chargement du résultat dans le registre D*

Notation : $D + M(R) \rightarrow D$

Cette nouvelle opération nécessite l'intervention supplémentaire d'un additionneur Σ et l'on est conduit à l'ensemble de la figure 18 où l'on retrouve les registres A, R, D et la mémoire M avec les entrées de commande déjà rencontrées.

L'additionneur Σ est avant tout un système combinatoire qui élabore en permanence la somme des mots binaires présents sur ses deux entrées A et B, c'est-à-dire la somme du contenu du registre D et du mot binaire présent sur le bus des données. Néanmoins, la sortie est munie d'un registre interne qui joue strictement le rôle de tampon du registre T de l'exemple du paragraphe II, 2, et qui charge la somme élaborée par l'additionneur au moment de la transition 0 à 1 à $(CI)_\Sigma$. Par ailleurs, la sortie de ce registre est reliée à l'entrée effective du registre D par un amplificateur trois-états interne à l'additionneur, « transparent » lorsque $(\overline{RD})_\Sigma$ est porté à



	$(\overline{RD})_R$	$(\overline{WR})_R$	$(CI)_R$		$(\overline{RD})_M$	$(\overline{WR})_M$	$(CI)_A$	Commentaire :
	1	1	0	—	1	1	0	
M(R) → R	1	1	0		1	1	1	Adressage de la mémoire par le contenu du registre R
	1	0	0		0	1	0	Réalisation des liaisons au bus des données
	1	0	1		0	1	0	Chargement du registre R
	1	1	0		1	1	0	

$M(R) \rightarrow R$

Fig. 17

l'état logique 0 et qui « isole » le registre de sortie de l'additionneur de l'entrée du registre D lorsque $(\overline{RD})_\Sigma = 1$. Dans toutes les opérations où l'additionneur Σ n'est pas utilisé (en particulier lorsque $(\overline{WR})_D = 0$ où c'est alors une obligation), on maintient $(\overline{RD})_\Sigma$ au niveau logique 1.

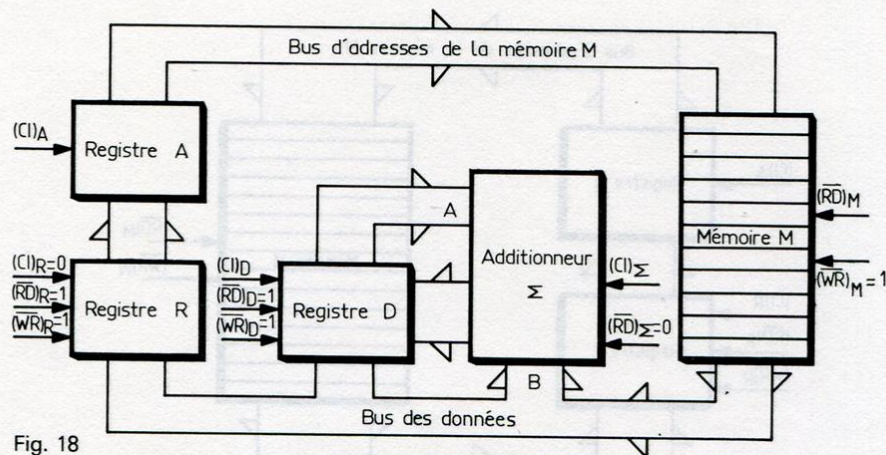


Fig. 18

	$(CI)_D$	$(RD)_M$	$(WR)_M$	$(CI)_A$	$(CI)_\Sigma$	Commentaire:
	1	1	1	0	0	
	0	1	1	0	0	
	0	1	1	1	0	Adressage de la mémoire par le contenu de R
	0	0	1	0	0	Liaison de la ligne pointée avec le bus des données
	0	0	1	0	1	Transfert du résultat à l'entrée de D
	1	1	1	0	0	Chargement de D
	1	1	1	1	0	

Fig. 19

$$D + M(R) \rightarrow D$$

Tout au cours de l'opération $D + M(R) \rightarrow D$, le registre R et le registre D sont déconnectés du bus des données grâce à $(RD)_R = 1$ et $(WR)_R = 1$ d'une part et $(RD)_R = 1$ et $(WR)_R = 1$ d'autre part. Il n'y a pas à charger le registre R : $(CI)_R$ demeure à 0.

La première phase de cette opération consiste à adresser la mémoire M qui peut alors être isolée du bus des données ($(\overline{RD})_M = 1$ et $(\overline{WR})_M = 1$) en chargeant le registre A par le contenu de R (passage de $(CI)_A$ de 0 à 1). Il faut ensuite relier la sortie de la ligne pointée au bus des données en faisant $(\overline{RD})_M = 0$. La somme $D + M(R)$ alors élaborée par l'additionneur Σ est chargée sur son registre de sortie en provoquant le passage de 0 à 1 de $(CI)_\Sigma$ et se retrouve donc sur l'entrée du registre D car $(\overline{RD})_\Sigma = 0$. Il ne reste plus qu'à charger le registre D par cette somme en faisant passer $(CI)_D$ de 0 à 1. La figure 19 représente la partie utile du contenu de la mémoire de l'automate qui commande cette séquence.

e) Incréméntation et décrémentation du registre R

Notation symbolique : $R + 1$ et $R - 1$

Cette incréméntation (ou cette décrémentation) ne se fait pas directement sur le registre R dont la structure ne le permet pas. Elle se fait par l'intermédiaire d'un circuit d'incréméntation - décrémentation qui peut fort bien être le compteur C de l'exemple du paragraphe II, 1, admettant les grandeurs de commande L (chargement), C/\overline{D} (comptage - décomptage) et $(CI)_C$ (horloge) (fig. 20). Nous ferons l'hypothèse supplémentaire (indispensable dans le paragraphe III - 2 qui suit) que la sortie de ce compteur est reliée à l'entrée effective du registre R par une logique trois états commandée par $(\overline{RD})_C$: lorsque $(\overline{RD})_C = 0$, le nombre présent en sortie du compteur est aussi présent sur l'entrée effective du registre R. $(\overline{RD})_C = 1$ « isole » le compteur de l'entrée du registre R (bien entendu, cette condition était implicitement réalisée au cours des opérations que nous avons étudiées précédemment, en particulier chaque fois que $(\overline{WR})_R = 0$).

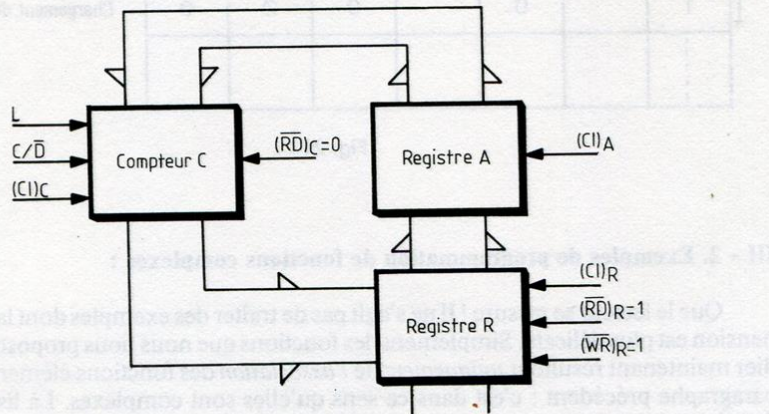


Fig. 20

Dans les opérations d'incréméntation et de décrémentation, le registre R reste isolé du bus des données : $(\overline{RD})_R = 1$ et $(\overline{WR})_R = 1$.

Prenons l'exemple de l'incréméntation du registre R. Le contenu initial du registre R est d'abord transféré sur le registre A au moment de la transition 0 à 1 de $(CI)_A$ en même temps que sur le compteur C alors « transparent » car $L = 1$. On doit ensuite incréménter le compteur C d'une unité ce qui est réalisé (en amenant L à 0 et C/\overline{D} à 1 (comptage) au moment de la transition 0 à 1 de $(CI)_C$. Le nouveau contenu du compteur C, présent à l'entrée effective du registre R doit alors être chargé dans le registre R en faisant passer $(CI)_R$ de 0 à 1. (voir contenu de la mémoire de l'automate en figure 21).

La décrémentation du registre R est réalisée de la même manière en amenant au moment voulu C/\overline{D} à 0 (décomptage) au lieu de 1 (comptage).

	$(CI)_R$		$(CI)_A$		L	C/\overline{D}	$(CI)_C$	Commentaire :
	⋮							
	Q	-----	0	-----	1	Ø	0	
	0		1		1	Ø	0	Chargement du registre A et du compteur C
R + 1	0		0		0	1	0	
	0		0		0	1	1	Incréméntation du compteur C
	1		0		0	Ø	0	Chargement du registre R
	⋮							

Fig. 21

III - 2. Exemples de programmation de fonctions complexes :

Que le lecteur se rassure ! Il ne s'agit pas de traiter des exemples dont la compréhension est plus délicate. Simplement, les fonctions que nous nous proposons d'étudier maintenant résultent *uniquement* de l'association des fonctions élémentaires du paragraphe précédent : c'est dans ce sens qu'elles sont complexes. La liste de ces fonctions élémentaires qui peuvent être utilisées forment un ensemble que nous appellerons « jeu d'instructions ».

Le système à commander est représenté sur la figure 22. Il résulte de l'association de tous les éléments de circuit utilisés au paragraphe III. 1. organisés autour d'un même *bus des données*. Les entrées de commande de ce système sont au nombre de 15. L'automate qui en réalisera la commande doit donc comprendre une mémoire d'au moins 15 colonnes. Les flèches portées sur les différents bus indiquent les sens dans lesquelles des mots binaires *peuvent* être véhiculés.

a) incrémentation et décrémentation du registre D

Ces deux opérations se traitent de la même façon. Prenons l'exemple de l'incrément du registre D (notation : $D + 1$)

Elle peut se faire en procédant, successivement aux étapes suivantes :

- chargement du contenu du registre D dans le registre R ($D \rightarrow R$) : paragraphe III. 1. a
- incrémentation du registre R ($R + 1$) : paragraphe III. 1. e)
- chargement du contenu du registre R dans le registre D ($R \rightarrow D$) : paragraphe III. 1. a

Les trois fonctions élémentaires utilisées ne font appel ni à la mémoire M qui doit demeurer « isolée » du bus des données ($(\overline{RD})_M = 1$ et $(\overline{WR})_M = 1$), ni à l'additionneur Σ qui doit rester « isolé » de l'entrée effective du registre D ($(\overline{RD})_\Sigma = 1$). $(C1)_\Sigma$ peut demeurer à l'état 0.

La partie utile du contenu de la mémoire de l'automate apparaît sur la figure 23. C'est avant tout une juxtaposition des contenus de la mémoire correspondant aux trois fonctions élémentaires utilisées.

b. *Programmation de la fonction* : Addition du nombre binaire N_1 contenu dans la mémoire M à l'adresse $a + 1$ et du nombre binaire N_2 contenu dans la mémoire M à l'adresse $a + 2$ et stockage du résultat N_0 dans la mémoire M à l'adresse a. On suppose que le registre R contient initialement le nombre a.

Les différentes étapes de la réalisation de cette fonction sont :

- incrémentation du registre R ($R + 1$: paragraphe III. 1. e) dont le contenu devient alors $a + 1$.
- chargement du contenu de la ligne de la mémoire M pointée par le registre R dans le registre D ($(M(R) \rightarrow D$: paragraphe III. 1. b). Le nombre binaire N_1 placé à l'adresse $a + 1$ est ainsi chargé dans le registre D.
- incrémentation du registre R ($R + 1$) dont le contenu devient $a + 2$.
- addition du contenu du registre D (soit N_1) et du contenu de la ligne de la mémoire M pointée par le registre R (soit N_2), et chargement du résultat (soit N_0) dans le registre D ($D + M(R) \rightarrow D$: paragraphe III. 1. d)

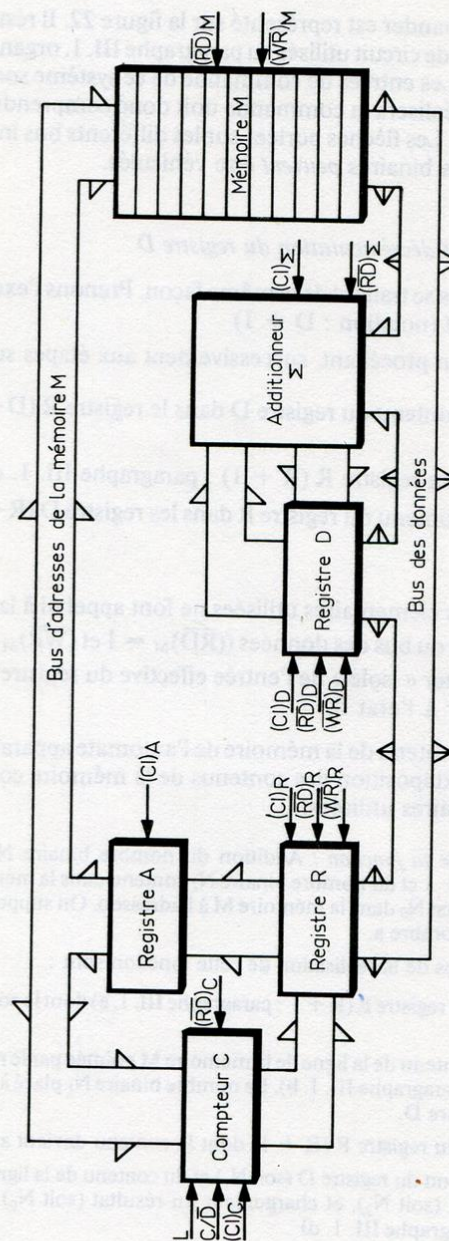


Fig. 22

$(RD)_R$	$(WR)_R$	$(CI)_R$	$(RD)_D$	$(WR)_D$	$(CI)_D$		$(CI)_A$		L	C/\bar{D}	$(C')_C$	$(RD)_C$

1	0	0	0	1	0	---	0	---	0	\emptyset	0	1
1	0	1	0	1	0		0		0	\emptyset	0	1
1	1	0	1	1	0		0		1	\emptyset	0	1
1	1	0	1	1	0		1		1	\emptyset	0	0
1	1	0	1	1	0		0		0	1	0	0
1	1	0	1	1	0		0		0	1	1	0
1	1	1	1	1	0		0		0	\emptyset	0	0
0	1	0	1	0	0		0		0	\emptyset	0	1
0	1	0	1	0	1		0		0	\emptyset	0	1

$\underbrace{\quad\quad\quad}_{D \rightarrow R}$
 $\underbrace{\quad\quad\quad}_{R+1}$
 $\underbrace{\quad\quad\quad}_{R \rightarrow D}$

Fig. 23

		$(\overline{RD})_R$	$(\overline{WR})_R$	$(CI)_R$	$(\overline{RD})_D$	$(\overline{WR})_D$	$(CI)_D$	$(\overline{RD})_M$	$(\overline{WR})_M$	$(CI)_A$	$(CI)_\Sigma$	$(\overline{RD})_\Sigma$	L	C/\overline{D}	$(CI)_C$	$(\overline{RD})_C$
R + 1	{	1	1	0	1	1	0	1	1	0	0	1	1	\emptyset	0	1
		1	1	0	1	1	0	1	1	1	0	1	1	\emptyset	0	0
		1	1	0	1	1	0	1	1	0	0	1	0	1	0	0
		1	1	0	1	1	0	1	1	0	0	1	0	1	1	0
		1	1	1	1	1	0	1	1	0	0	1	0	\emptyset	0	0
M(R) \rightarrow D	{	1	1	0	1	1	0	1	1	0	0	1	0	\emptyset	0	1
		1	1	0	1	1	0	1	1	1	0	1	0	\emptyset	0	1
		1	1	0	1	0	0	0	1	0	0	1	0	\emptyset	0	1
		1	1	0	1	0	1	0	1	0	0	1	0	\emptyset	0	1
R + 1	{	1	1	0	1	1	0	1	1	0	0	1	1	\emptyset	0	1
		1	1	0	1	1	0	1	1	1	0	1	1	\emptyset	0	0
		1	1	0	1	1	0	1	1	0	0	1	0	1	0	0
		1	1	0	1	1	0	1	1	0	0	1	0	1	1	0
		1	1	1	1	1	0	1	1	0	0	1	0	\emptyset	0	0
D + M(R) \rightarrow D	{	1	1	0	1	1	0	1	1	0	0	0	0	\emptyset	0	1
		1	1	0	1	1	0	1	1	1	0	0	0	\emptyset	0	1
		1	1	0	1	1	0	0	1	0	0	0	0	\emptyset	0	1
		1	1	0	1	1	0	0	1	0	1	0	0	\emptyset	0	1
		1	1	0	1	1	1	1	1	0	0	0	1	\emptyset	0	1
R - 1	{	1	1	0	1	1	0	1	1	0	0	1	1	\emptyset	0	1
		1	1	0	1	1	0	1	1	1	0	1	1	\emptyset	0	0
		1	1	0	1	1	0	1	1	0	0	1	0	0	0	0
		1	1	0	1	1	0	1	1	0	0	1	0	0	1	0
		1	1	0	1	1	0	1	1	0	0	1	0	\emptyset	0	0
R - 1	{	1	1	0	1	1	0	1	1	0	0	1	1	\emptyset	0	1
		1	1	0	1	1	0	1	1	1	0	1	1	\emptyset	0	0
		1	1	0	1	1	0	1	1	0	0	1	0	0	0	0
		1	1	0	1	1	0	1	1	0	0	1	0	0	1	0
		1	1	0	1	1	0	1	1	0	0	1	0	\emptyset	0	0
D \rightarrow M(R)	{	1	1	0	1	1	0	1	1	0	0	1	0	\emptyset	0	1
		1	1	0	1	1	0	1	1	1	0	1	0	\emptyset	0	1
		1	1	0	0	1	0	1	0	0	0	1	0	\emptyset	0	1

Fig. 24

- décrémentation du registre R ($R - 1$: paragraphe III. 1. e) dont le contenu devient $a - 1$
- nouvelle décrémentation du registre R ($R - I$) pour lui faire supporter le nombre a
- stockage du contenu du registre D (soit N_0) dans la ligne de la mémoire M pointée par le registre R ($D \rightarrow M(R)$: paragraphe III. 1. b) qui amène N_0 dans la ligne mémoire d'adresse a.

On se rend compte que le chargement de la mémoire de l'automate (programmation) destiné à commander le système (voir figure 24) est une opération longue et fastidieuse même pour la réalisation d'une fonction qui, comme c'est le cas ici, n'est pas très compliquée.

Chapitre 3

AUTOMATE PROGRAMMABLE COMPOSÉ

L'automate programmable qui a été décrit dans le chapitre précédent permet théoriquement d'effectuer le séquençement de n'importe quelle suite de micro-instructions.

De ce point de vue, il est doué d'une certaine universalité et présente un grand intérêt pour la commande de nombreux systèmes automatiques.

Cependant, nous nous sommes rendu compte que la programmation d'une fonction dans la mémoire de l'automate, bien qu'aisée à comprendre, peut devenir une opération longue et fastidieuse. Cette opération se justifie néanmoins si l'automate est destiné à *remplir toujours la même mission* (par exemple commander le système permettant d'effectuer $x + 3y$, ou bien fournir les microactions nécessaires à la réalisation de la fonction décrite au paragraphe III. 2. b du chapitre précédent) : la programmation est faite une fois pour toute.

Dans bien des cas, il serait plus commode de disposer d'un automate capable d'effectuer sur un système la commande d'un certain nombre de fonctions élémentaires, l'utilisateur ayant pour seule tâche de programmer *l'ordre* dans lequel il souhaite que ces fonctions élémentaires soit associées pour former une fonction complexe.

Nous allons donc présenter un nouvel automate, formé de deux automates programmables, qui fonctionne suivant le principe qui vient d'être énoncé. Nous nous attacherons à montrer, à l'aide de quelques exemples, l'intérêt d'un tel dispositif par rapport à un automate simple.

I. Description et fonctionnement du système

I. 1. Présentation du système

Considérons le système de la figure 1, réalisé en associant deux automates programmables. Le compteur de l'automate A s'incrémente lors d'un front montant du signal d'horloge appliqué sur l'entrée de commande Cl_A . Le compteur de l'automate B s'incrémente lors d'un front montant du signal appliqué sur l'entrée de commande Cl_B . Ce signal est fourni par un bit de sortie de la mémoire de l'automate A.

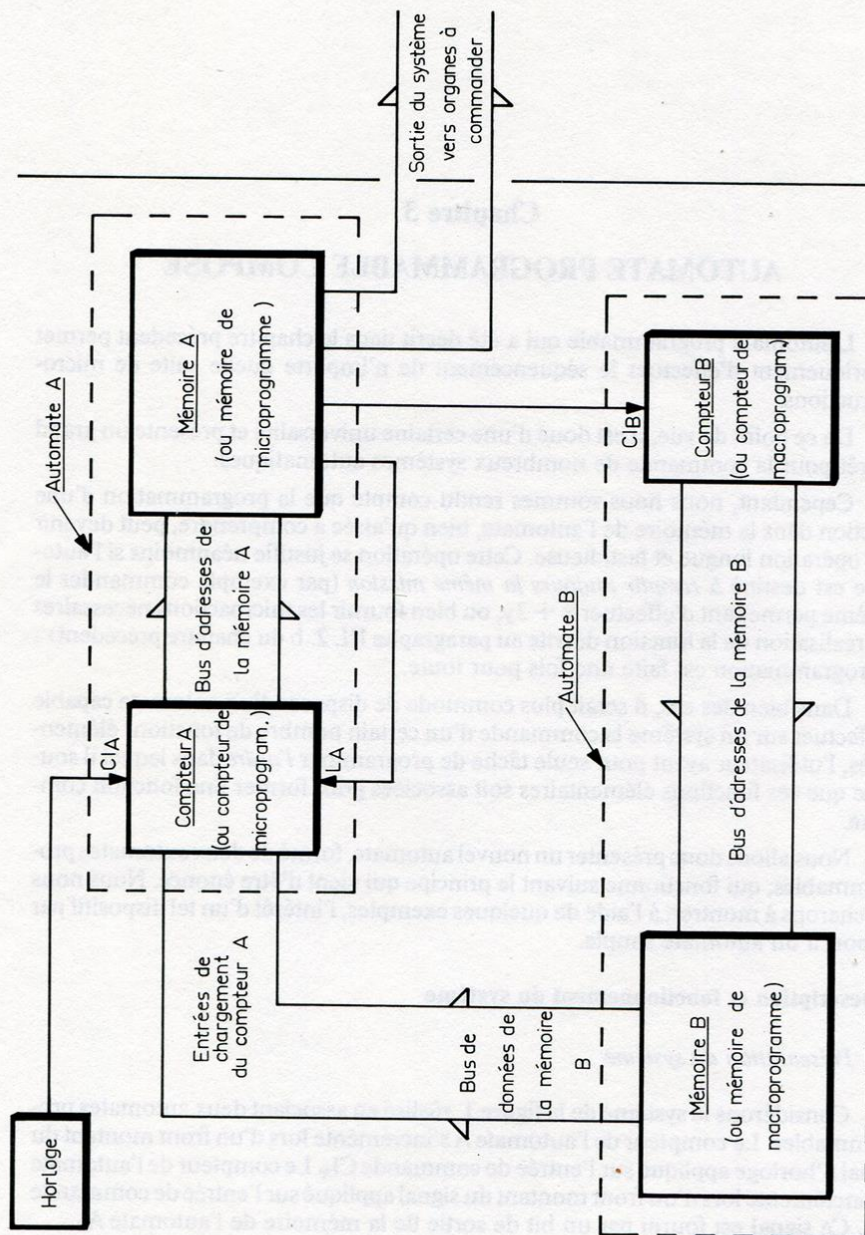


Fig. 1. Automate programmable composé

Un niveau logique 1 appliqué sur l'entrée de commande L_A , délivré par un bit de sortie de la mémoire de l'automate A rend « transparent » le compteur de l'automate A qui alors ne peut s'incrémenter lors d'un front montant de Cl_A ; rappelons que dans ce cas, le nombre binaire en sortie du compteur est égal au nombre binaire présent à son entrée.

Les mémoires A et B sont adressées respectivement par les compteurs A et B.

Parmi les bits de sortie de la mémoire A, deux sont utilisés pour la commande d'organes du système lui-même (L_A , Cl_B). Les autres sont disponibles pour la commande d'organes extérieurs et constituent la sortie effective du système « automate programmable composé ».

I. 2. Contenu de la mémoire de l'automate A

Dans la mémoire de l'automate A sont programmées, en vue de l'utilisation du système complet, un certain nombre de séquences de *microinstructions*, chaque séquence correspondant à une *macroinstruction* (ou plus simplement *instruction*) à réaliser (*exécuter*). Par exemple, dans la mémoire A, de la ligne n_i à la ligne $n_j - 1$ sont programmées les microinstructions dont l'exécution correspond à la réalisation de l'instruction « incrémentation du registre R ». De la ligne n_j à la ligne $n_k - 1$ peuvent être programmées les microinstructions dont l'exécution conduit au « chargement du contenu du registre R dans le registre D qui constitue une autre macroinstruction (fig. 2)

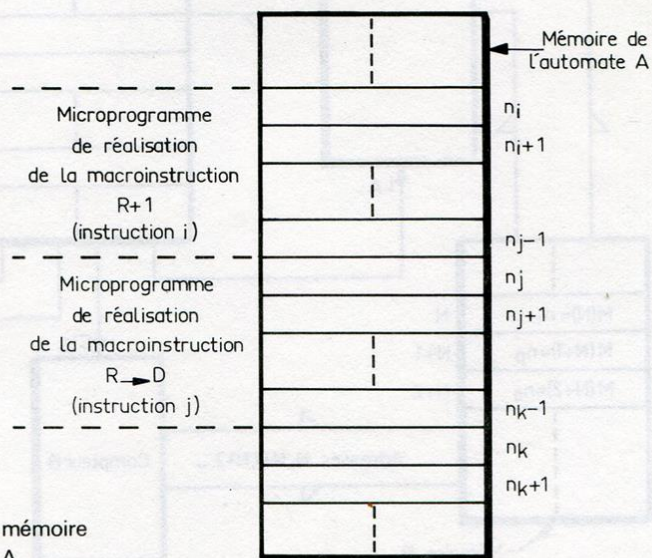


Fig. 2. Contenu de la mémoire de l'automate A

I. 3. Finalité du système

Le but que l'on poursuit est l'exécution d'une séquence déterminée de ces macroinstructions, *dans l'ordre que l'on désire* (par exemple instruction i, puis instruction l, puis instruction k, puis à nouveau instruction i...) et pas forcément dans l'ordre où ces macroinstructions ont été programmées dans la mémoire de l'automate A (instruction i, puis instruction j, puis k...).

Soit n_i (respectivement n_j , n_k ...) l'adresse de la première ligne de l'instruction i (respectivement j, k...) dans la mémoire de l'automate A. L'ordre dans lequel on

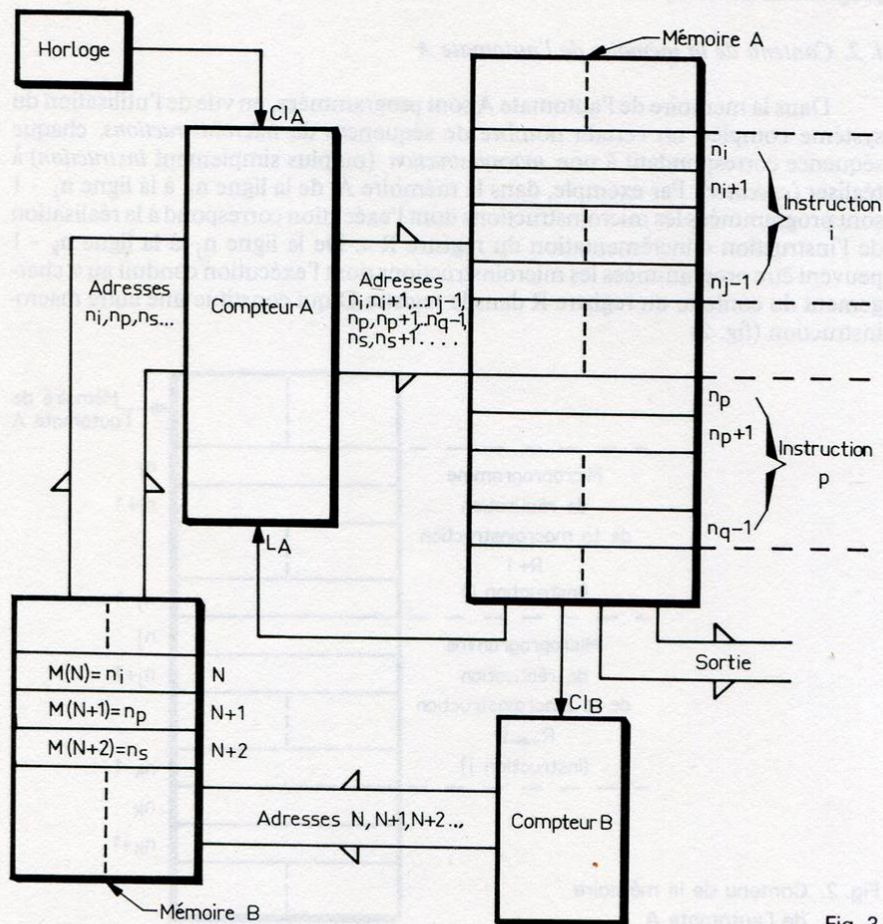


Fig. 3

désire que ces macroinstructions soient réalisées est enregistré dans la mémoire B sous forme des adresses successives n_i ou n_j ou $n_k \dots$ caractéristiques de ces instructions.

I. 4. Fonctionnement général

On peut suivre l'exposé du fonctionnement du système sur la figure 3 où la mémoire B est supposée en fonction lecture. La figure 3 bis précise quelques situations.

a) Le compteur de l'automate B fournit une adresse N à la mémoire B. Le mot $M(N)$ contenu dans la ligne d'adresse N de la mémoire B est l'adresse n_i de la première des lignes de la mémoire A où est enregistré le microprogramme de l'instruction i .

b) $M(N) = n_i$ est chargé dans le compteur A et la ligne n_i de la mémoire A est effectivement pointée.

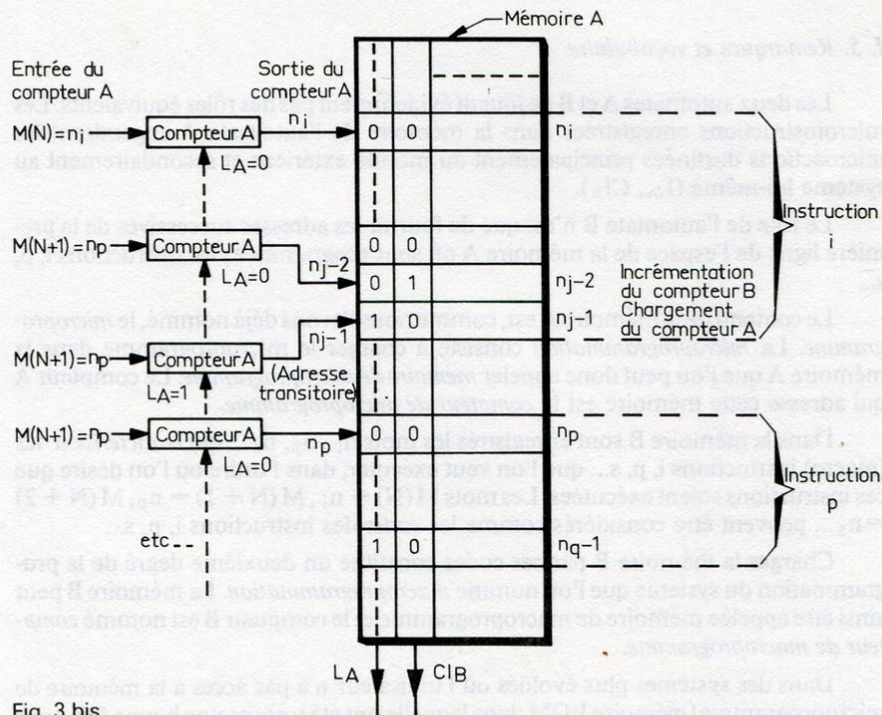


Fig. 3 bis

c) le microséquencement dû aux incréments successives du compteur A démarre à l'adresse n_i de la mémoire A et engendre l'exécution des microinstructions qui constituent la macroinstruction i .

d) quelle que soit la nature de la macroinstruction i , le microprogramme élémentaire qui permet l'exécution de i comporte une microinstruction d'incrémenta-tion du compteur B qui passe donc à $N + 1$.

e) Le mot $M(N + 1)$ contenu dans la ligne d'adresse $N + 1$ de la mémoire B est l'adresse n_p de la première des lignes de la mémoire A où est enregistré le microprogramme de la macroinstruction suivante p . $M(N + 1) = n_p$ est chargé dans le compteur A et la ligne n_p est effectivement pointée.

f) Le microséquencement dû aux incréments successives du compteur A reprend mais cette fois à l'adresse n_p et engendre l'exécution de l'instruction p .

g) Le microprogramme de l'instruction p comporte lui aussi une incrémentation du compteur B qui pointe alors la ligne d'adresse $N + 2$ où est enregistrée l'adresse n_s de la première des lignes de la mémoire A où est programmée l'instruction s , et la même procédure reprend.

I. 5. Remarques et vocabulaire

Les deux automates A et B ne jouent évidemment pas des rôles équivalents. Les microinstructions enregistrées dans la mémoire de l'automate A engendrent les microactions destinées principalement au monde extérieur et secondairement au système lui-même (L_A , Cl_B).

Le rôle de l'automate B n'est que de fournir les adresses successives de la première ligne de l'espace de la mémoire A où sont programmées les instructions i , p , s ...

Le contenu de la mémoire A est, comme nous l'avons déjà nommé, le *microprogramme*. La *microprogrammation* consiste à charger le microprogramme dans la mémoire A que l'on peut donc appeler *mémoire de microprogramme*. Le compteur A qui adresse cette mémoire est le *compteur de microprogramme*.

Dans la mémoire B sont enregistrés les mots n_i , n_p , n_s ... qui *caractérisent* les (macro) instructions i , p , s ... que l'on veut exécuter, dans l'ordre où l'on désire que ces instructions soient exécutées. Les mots $M(N) = n_i$, $M(N + 1) = n_p$, $M(N + 2) = n_s$... peuvent être considérés comme les *codes* des instructions i , p , s ...

Charger la mémoire B par ces codes constitue un deuxième degré de la programmation du système que l'on nomme *macroprogrammation*. La mémoire B peut ainsi être appelée *mémoire de macroprogramme* et le compteur B est nommé *compteur de macroprogramme*.

Dans des systèmes plus évolués où l'utilisateur n'a pas accès à la mémoire de microprogramme (mémoire ROM dans laquelle ont été inscrits une bonne fois pour

toutes les microprogrammes de certaines instructions bien définies) ni au compteur de microprogramme, la seule programmation à effectuer est la macroprogrammation. Les mémoire et compteur A sont alors plus simplement nommés *mémoire et compteur de programme*.

II. Exemples d'utilisation

II. 1. Commande d'un compteur

Comme dans le chapitre 2 (paragraphe II. 1), on désire commander, à l'aide de l'automate composé, un compteur C et faire exécuter la séquence, dans l'ordre indiqué : chargement, incrémentation, incrémentation, décrémentation, décrémentation, en agissant sur les entrées de commande L , $(Cl)_C$ et C/\bar{D} .

a) microprogrammation

La séquence que l'on veut faire exécuter est une combinaison des trois fonctions élémentaires : chargement, incrémentation, décrémentation du compteur C. Chacune de ces instructions doit être microprogrammée dans la mémoire A.

Intéressons-nous d'abord au microprogramme de l'instruction « chargement du compteur C », enregistré dans la mémoire A à partir de la ligne d'adresse n_i . Ce microprogramme se décompose en deux parties :

- microprogramme correspondant effectivement à l'instruction « chargement du compteur C » se réduisant à une seule ligne, la ligne n_i (il suffit d'imposer $L = 1$).
- microprogramme permettant l'incrément du compteur B (passage de Cl_B de 0 à 1 réalisé grâce à la ligne $n_i + 1$) et le chargement dans le compteur A de la nouvelle donnée de la mémoire B (passage de L_A à 1, ligne $n_i + 2$).

La première partie de ce microprogramme est destinée à produire les microactions nécessaires à la réalisation de l'instruction : c'est le microprogramme *d'exécution* de l'instruction en cours.

La seconde partie est de nature différente. Elle concerne la commande de l'automate lui-même. Lors de son exécution, l'automate reconnaît l'instruction suivante à traiter dont le code est inscrit dans la mémoire B et le compteur A se prépare à pointer la première ligne de l'espace de la mémoire A où est enregistré le microprogramme de l'instruction suivante. C'est la partie de microprogramme de *recherche* de l'instruction suivante.

Le microprogramme de l'instruction « incrément du compteur C », enregistré à partir de l'adresse n_k et le microprogramme de l'instruction « décrémentation

tion du compteur C » enregistré à partir de la ligne n_p comportent également ces deux phases. On se souvient par ailleurs que l'incréméntation (ou la décrémentation) du compteur C se fait en imposant d'abord $L=0$ et $C/\bar{D}=1$ (ou respectivement 0) puis en provoquant le passage de $(Cl)_C$ de 0 à 1 (fig. 4).

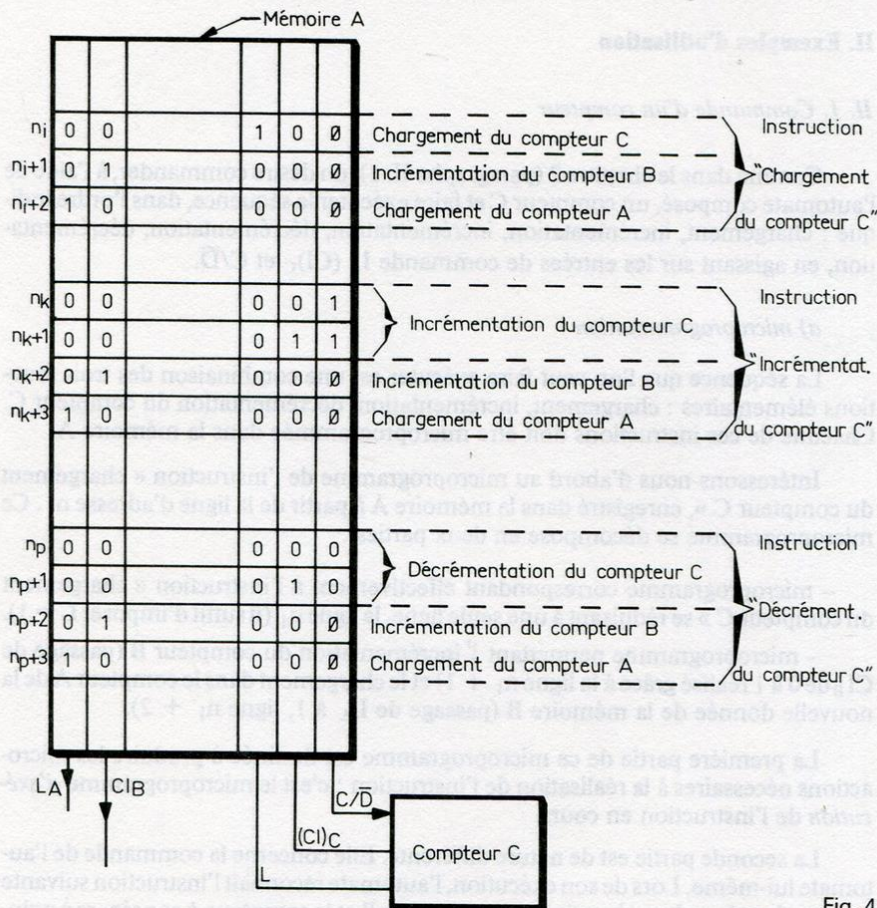


Fig. 4

b) macroprogrammation

Les adresses n_i , n_k et n_p de début des microprogrammes de chargement, incréméntation et décrémentation du compteur C sont les *codes* de ces instructions.

La macroprogrammation consiste à écrire dans la mémoire B, dans l'ordre où l'on veut les voir exécuter, les codes des trois instructions, soit n_i , n_k , n_p , n_p . On terminera par le code n_s d'une instruction d'arrêt du microséquencement. La réalisation de cette fonction peut consister en la neutralisation de l'horloge qui fournit Cl_A lorsqu'un bit T passe à 0 grâce à l'emploi d'une cellule « Et » (fig. 5).

c) déroulement du programme

L'automate composé et le contenu des mémoires est représenté sur la figure 5. On suppose que les deux compteurs A et B de l'automate ont été remis à 0 : chacun pointe la ligne d'adresse 0 de la mémoire correspondante.

1. Dès la mise en route du système (mise en marche de l'horloge), le compteur A se charge par n_i inscrit dans la mémoire B à l'adresse 0 (grâce au contenu de lignes 0 et 1 de la mémoire A).
2. L'instruction microprogrammée à partir de n_i dans la mémoire A (chargement du compteur C) s'exécute.
3. Le compteur B est incrémenté et la ligne 1 de la mémoire B est pointée.
4. Le compteur A se charge par n_k inscrit dans la mémoire B à l'adresse 1 et la ligne n_x de la mémoire A est pointée.
5. L'instruction microprogrammée à partir de l'adresse n_k dans la mémoire A (incrémentement du compteur C) s'exécute.
6. Le compteur B est incrémenté et la ligne 2 de la mémoire B est pointée.
7. Le compteur A se charge à nouveau par n_k inscrit dans la mémoire B à l'adresse 2.
8. L'instruction « incrémentement » s'exécute à nouveau.
9. Le compteur B est incrémenté et la ligne 3 de la mémoire B est pointée.
10. Le compteur A se charge par n_p inscrit dans la mémoire B à l'adresse 3.
11. L'instruction microprogrammée à partir de l'adresse n_p dans la mémoire A (décrémentement du compteur C) s'exécute.
12. Le compteur B est incrémenté et la ligne 4 de la mémoire B est pointée.
13. Le compteur A se charge à nouveau par n_p inscrit dans la mémoire B à l'adresse 4.
14. L'instruction « décrémentement » s'exécute à nouveau.
15. Le compteur B est incrémenté et la ligne 5 de la mémoire B est pointée.
16. Le compteur A se charge par n_s inscrit dans la mémoire B à l'adresse 5.
17. L'arrêt du microséquencement s'exécute grâce au passage de T à 0.

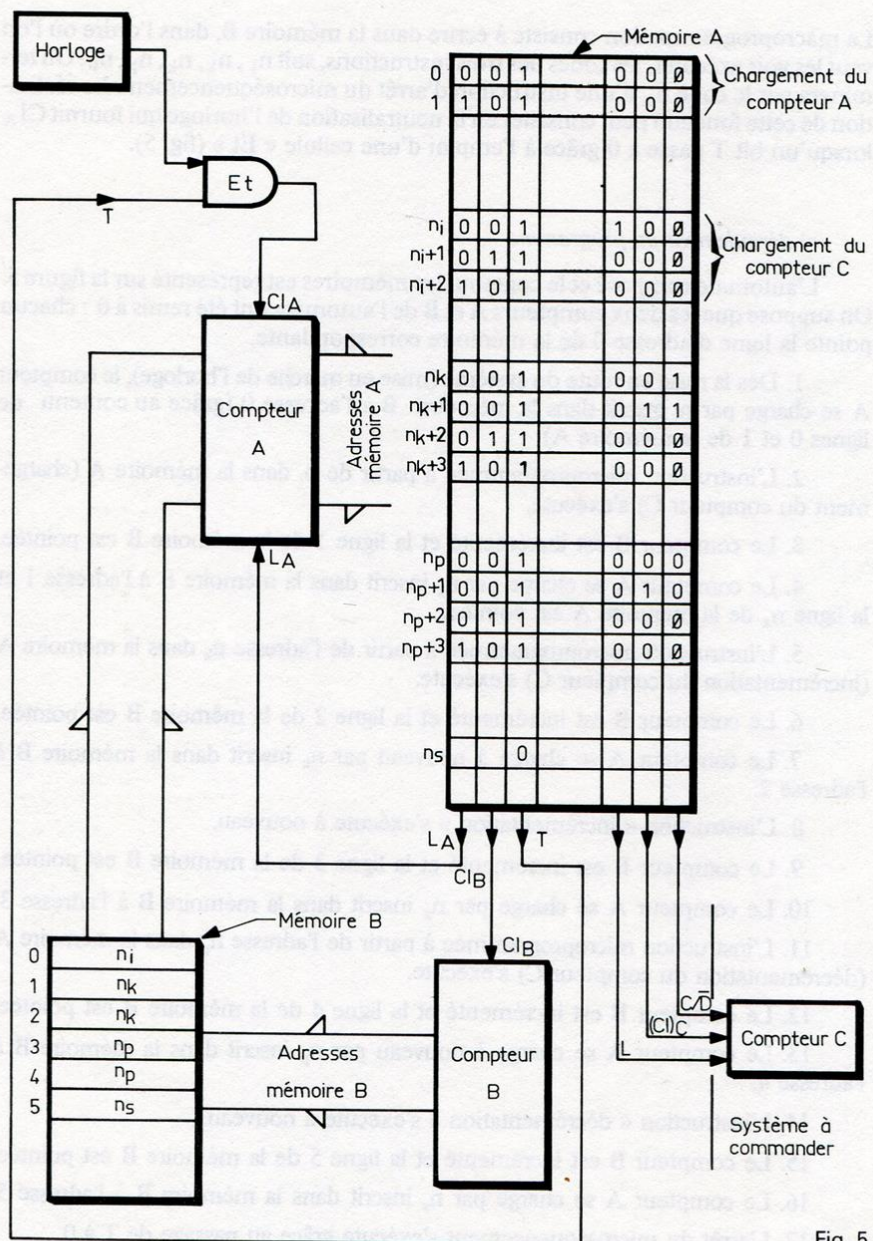


Fig. 5

d) Remarques

1. Cette longue énumération met en évidence que le déroulement du programme consiste en une alternance des deux phases :

- *exécution* d'une instruction
- *recherche* de l'instruction suivante à exécuter (par l'incrémentation du compteur B et le chargement du compteur A)

L'automate combiné fonctionne donc suivant le cycle :



2. On peut penser que la programmation complète de l'automate (microprogrammation et macroprogrammation) est somme toute une opération beaucoup plus longue que la programmation d'un automate simple. Mais il s'agit là d'un faux problème. Il faut en fait supposer que la microprogrammation des instructions élémentaires est effectuée, l'utilisateur n'ayant qu'à faire la macroprogrammation qui est beaucoup plus courte. Ce qui fait l'intérêt de cet automate composé, c'est que l'on peut lui demander d'effectuer une autre fonction complexe, combinaison des mêmes fonctions élémentaires, en ne reprenant que la macroprogrammation. Par exemple, si l'on veut affecter à l'automate la mission de commander, sur le compteur C, la séquence « chargement - décrémentation - incrémentation - décrémentation - incrémentation », il *suffit* de modifier le contenu de la mémoire de programme B en chargeant dans l'ordre, à partir de l'adresse 0 les codes n_i , n_p , n_k , n_p , n_k , puis le code n_s pour arrêter le microséquencement.

On conçoit que si l'on multiplie le nombre des fonctions élémentaires que l'automate peut effectuer, on peut en les composant dans des ordres différents obtenir des fonctions complexes très variées.

II. 2. Commande d'un système permettant d'effectuer $ax + by + cz$ (a, b et c entiers relatifs)

Le système en question (fig. 6) comprend :

- trois registres R, T et D qui peuvent être chargés dans le mode parallèle au moment de la transition 0 à 1 du signal Cl_X ($X = R$ ou T ou D). En outre, le registre D peut être remis à 0, au moment de la transition de 0 à 1 de Cl_D si R est au niveau logique 1.

- deux multiplexeurs permettant d'aiguiller vers les entrées de chargement parallèle du registre R, soit x, soit y, soit z. Si $M_2 = 1$, quel que soit l'état de M_1 , $E_R = x$. Si $M_1 = 0$ et $M_2 = 0$, $E_R = y$. Si $M_1 = 1$ et $M_2 = 0$, $E_R = z$.

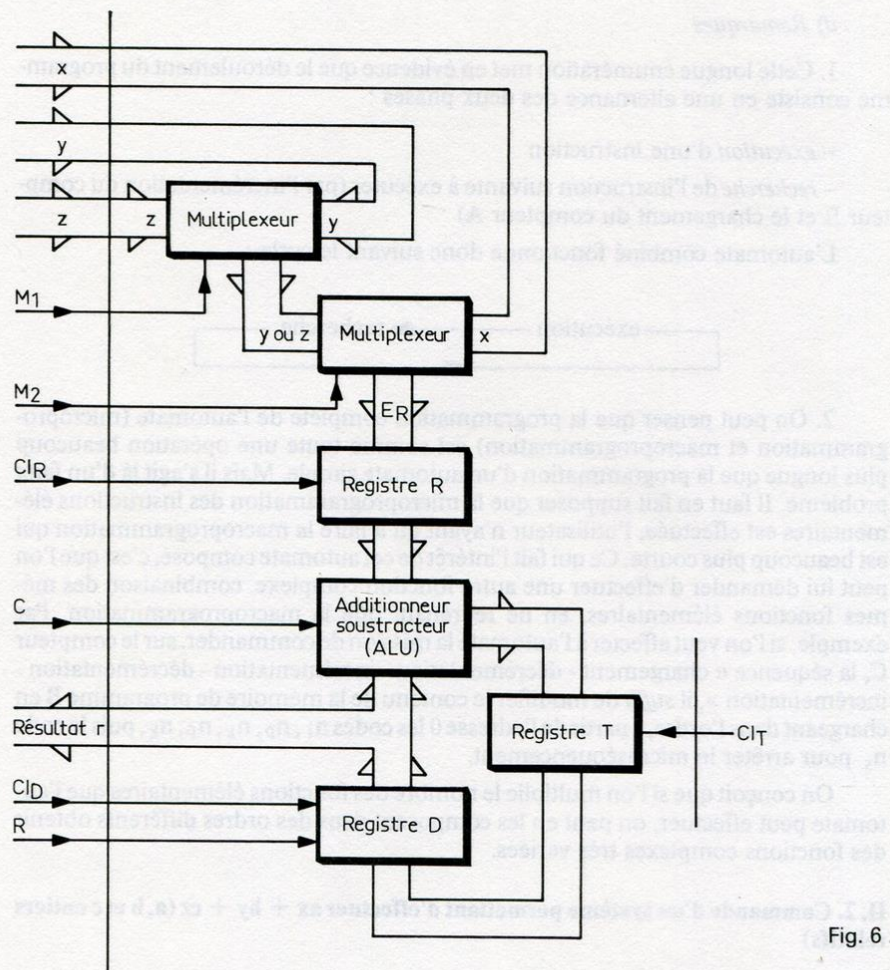


Fig. 6

– une unité arithmétique et logique (ALU) que l'on utilise uniquement pour effectuer deux opérations sélectionnées par la commande C. Si $C = 0$, l'unité additionne les contenus des registres R et D. Si $C = 1$, elle soustrait le contenu du registre R du contenu du registre D.

Les microactions à exercer sont au nombre de 7 : G , Cl_R , Cl_D , Cl_T , R , M_1 et M_2 auxquelles il faut ajouter les 3 microactions internes à l'automate composé : L_A , Cl_B et T .

a) microprogrammation

On se propose d'élaborer le microprogramme permettant l'exécution des (macro) instructions suivantes :

- chargement de x dans le registre R
- chargement de y dans le registre R
- chargement de z dans le registre R
- mise à 0 du registre D
- addition des contenus des registres R et D et transfert du résultat dans le registre D
- soustraction du contenu du registre R du contenu du registre D et transfert du résultat dans le registre D.

Si l'on décide – arbitrairement – de microprogrammer des instructions dans l'ordre où l'on vient de les citer, à partir de l'adresse binaire 00000 de la mémoire A, en n'oubliant pas que les deux premières lignes doivent être réservées à la programmation du chargement initial du compteur A, le contenu de la mémoire A peut être celui indiqué par la figure 7.

Pour chaque instruction, l'adresse de la première ligne a été encadrée : elle constitue le code de cette instruction qui sera utilisé dans la macroprogrammation chaque fois que l'on fera appel à cette instruction.

b) macroprogrammation

Proposons-nous par exemple de faire effectuer par le système de la figure 6 l'opération particulière $x + 2y - 2z$ (c'est à dire que l'on choisit le cas particulier $a = 1, b = 2, c = -2$).

La suite des opérations que l'automate devra faire exécuter est :

- mettre le registre D à 0
- charger x dans le registre R
- additionner les contenus des registres R (soit x) et D (soit 0) et charger le résultat (x) dans le registre D.
- charger y dans le registre R
- additionner les contenus des registres R (y) et D (x) et charger le résultat ($x + y$) dans le registre D
- additionner les contenus des registres R (y) et D ($x + y$) et charger le résultat ($x + 2y$) dans le registre D
- charger z dans le registre R

Adresses (binaire)	L _A	Cl _B	T	M ₁	M ₂	Cl _R	Cl _D	Cl _T	R	C	
0 0 0 0 0	0	0	1	∅	∅	0	0	0	∅	∅	Chargement initial du compteur A
0 0 0 0 1	1	0	1	∅	∅	0	0	0	∅	∅	
0 0 0 1 0	0	0	1	∅	1	0	0	0	∅	∅	Chargement de x dans le registre R
0 0 0 1 1	0	0	1	∅	1	1	0	0	∅	∅	
0 0 1 0 0	0	1	1	∅	∅	0	0	0	∅	∅	
0 0 1 0 1	1	0	1	∅	∅	0	0	0	∅	∅	Chargement de y dans le registre R
0 0 1 1 0	0	0	1	0	0	0	0	0	∅	∅	
0 0 1 1 1	0	0	1	0	0	1	0	0	∅	∅	
0 1 0 0 0	0	1	1	∅	∅	0	0	0	∅	∅	Chargement de z dans le registre R
0 1 0 0 1	1	0	1	∅	∅	0	0	0	∅	∅	
0 1 0 1 0	0	0	1	1	0	0	0	0	∅	∅	
0 1 0 1 1	0	0	1	1	0	1	0	0	∅	∅	Chargement de z dans le registre R
0 1 1 0 0	0	1	1	∅	∅	0	0	0	∅	∅	
0 1 1 0 1	1	0	1	∅	∅	0	0	0	∅	∅	
0 1 1 1 0	0	0	1	∅	∅	0	0	0	1	∅	Mise à 0 du registre D
0 1 1 1 1	0	0	1	∅	∅	0	1	0	1	∅	
1 0 0 0 0	0	1	1	∅	∅	0	0	0	∅	∅	
1 0 0 0 1	1	0	1	∅	∅	0	0	0	∅	∅	Addition des contenus des registres R et D
1 0 0 1 0	0	0	1	∅	∅	0	0	0	∅	0	
1 0 0 1 1	0	0	1	∅	∅	0	0	1	0	0	
1 0 1 0 0	0	0	1	∅	∅	0	1	0	0	∅	Résultat dans le registre D
1 0 1 0 1	0	1	1	∅	∅	0	0	0	∅	∅	
1 0 1 1 0	1	0	1	∅	∅	0	0	0	∅	∅	Soustraction du contenu du registre R du contenu du registre D
1 0 1 1 1	0	0	1	∅	∅	0	0	0	∅	1	
1 1 0 0 0	0	0	1	∅	∅	0	0	1	0	1	
1 1 0 0 1	0	0	1	∅	∅	0	1	0	0	∅	Résultat dans le registre D
1 1 0 1 0	0	1	1	∅	∅	0	0	0	∅	∅	
1 1 0 1 1	1	0	1	∅	∅	0	0	0	∅	∅	Arrêt du séquen- cement
1 1 1 0 0	0	0	0	∅	∅	0	0	0	∅	∅	

Fig. 7 : Contenu de la mémoire A.

- soustraire le contenu du registre R (z) du contenu du registre D ($x + 2y$) et charger le résultat ($x + 2y - z$) dans le registre D
- soustraire le contenu du registre R (z) du contenu du registre D ($x + 2y - z$) et charger le résultat ($x + 2y - 2z$) dans le registre D
- arrêter le séquençement.

Le macroprogramme à enregistrer dans la mémoire B est donné par la figure 8.

c) Remarque

L'automate ainsi programmé (microprogrammé et macroprogrammé) permet donc de commander la résolution de l'opération $x + 2y - 2z$.

Supposons que l'on veuille maintenant effectuer l'opération $-x - y + 3z$ en utilisant le même système de la figure 6. Il est évidemment inutile de reprendre la microprogrammation de l'automate composé. Nous avons seulement à modifier le macroprogramme enregistré dans la mémoire B dont le contenu pourra être celui de figure 9. Cette nouvelle programmation ne nécessite que le chargement de dix mots dans la mémoire B. On imagine aisément que la programmation de la même commande dans l'automate simple du chapitre 2 aurait nécessité l'écriture d'un nombre beaucoup plus important de mots dans la mémoire de cet automate.

Adresses (en binaire)							Instructions (écriture symbolique)
0 0 0 0	—	—	—	—	—	0	$0 \rightarrow D$
0 0 0 1						0	$x \rightarrow R$
0 0 1 0						0	$D + R \rightarrow D$
0 0 1 1						0	$y \rightarrow R$
0 1 0 0						0	$D + R \rightarrow D$
0 1 0 1						0	$D + R \rightarrow D$
0 1 1 0						0	$Z \rightarrow R$
0 1 1 1						0	$D - R \rightarrow D$
1 0 0 0						0	$D - R \rightarrow D$
1 0 0 1						0	Arrêt
						1	
						1	

Fig. 8. Contenu de la mémoire B pour effectuer $x + 2y - 2z$

0 0 0 0	- - - - -	0	0	1	1	1	0	$0 \rightarrow D$
0 0 0 1		0	0	0	0	1	0	$x \rightarrow R$
0 0 1 0		0	1	0	1	1	1	$D - R \rightarrow D$
0 0 1 1		0	0	0	1	1	0	$y \rightarrow R$
0 1 0 0		0	1	0	1	1	1	$D - R \rightarrow D$
0 1 0 1		0	0	1	0	1	0	$z \rightarrow R$
0 1 1 0		0	1	0	0	1	0	$D + R \rightarrow D$
0 1 1 1		0	1	0	0	1	0	$D + R \rightarrow D$
1 0 0 0		0	1	0	0	1	0	$D + R \rightarrow D$
1 0 0 1		0	1	1	1	0	0	Arrêt
		1						

Fig. 9 . Contenu de la mémoire B pour effectuer $-x - y + 3y$

III. Assimilation

Reprenons le système de figure 22 du chapitre 2. Nous nous sommes exercés à programmer dans la mémoire d'un automate simple la commande d'un certain nombre de fonctions élémentaires. Nous avons ensuite programmé des fonctions complexes en écrivant dans la mémoire de l'automate les programmes des fonctions élémentaires, dans l'ordre où ces fonctions élémentaires interviennent pour former ces fonctions complexes.

La commande de ces fonctions complexes – et d'autres fonctions complexes qui n'ont pas été envisagées au chapitre 2 – peut être avantageusement réalisée par un automate composé.

Rappelons que le système de la figure 22 est directement inspiré du microprocesseur COSMAC CDP 1802 et que le lecteur se familiarise ici avec d'une part une *structure* et d'autre part un *fonctionnement*.

III. 1. Microprogrammation

La microprogrammation des fonctions élémentaires a déjà été faite dans le chapitre précédent. Néanmoins, il ne faut pas oublier que le microprogramme de cha-

que instruction doit se terminer par les microinstructions de préparation à l'instruction suivante (incrémentement du compteur B et chargement du compteur A : phase de recherche).

Le contenu de la mémoire A est donné figure 10. L'ordre dans lequel les différentes instructions ont été programmées est absolument arbitraire. Les deux premières lignes commandent le chargement initial du compteur A. L'adresse de la première ligne de chaque instruction, indiquée en binaire, représente le code de cette instruction. Cette microprogrammation est longue mais elle n'est faite qu'une seule fois.

III. 2. Macroprogrammation

a) incrémentation et décrémentation du registre D ($D + 1$ et $D - 1$)

On a vu que le principe de l'incrémentement du registre D consiste à charger le registre R par le contenu du registre D ($D \rightarrow R$) puis incrémenter le registre R ($R + 1$) et enfin charger le registre D par le contenu du registre R ($R \rightarrow D$). La mémoire B contient, à partir de l'adresse 0, les codes de ces trois instructions successives. (fig. 11).

La décrémentation du registre D se fait de la même manière par l'intermédiaire de la décrémentation du registre R.

b) Addition des nombres binaires N_1 et N_2 contenus dans la mémoire M respectivement aux adresses $a + 1$ et $a + 2$ et stockage du résultat N_0 dans la mémoire M à l'adresse a , le registre R contenant initialement a .

La fonction complexe à réaliser résulte de l'exécution, dans l'ordre, des fonctions élémentaires $R + 1$, $M(R) \rightarrow D$, $R + 1$, $D + M(R) \rightarrow D$, $R - 1$, $R - 1$, $D \rightarrow M(R)$. D'où le contenu de la mémoire de macroprogramme B indiqué sur la figure 12. Sans compter l'instruction « arrêt », la programmation nécessite l'écriture de 7 mots de 8 bits (utilisation d'une mémoire standard) alors qu'avec un automate simple, il faut écrire 32 mots de 16 bits !

III. 3 Remarques

a) La simplification de la programmation entraînée par l'utilisation d'un automate composé (seule la macroprogrammation étant à effectuer) permet d'envisager la réalisation de fonctions beaucoup plus compliquées. L'assemblage des fonctions élémentaires pour former une fonction complexe déterminée est facilité par l'emploi de techniques (techniques de programmation) qu'il ne nous appartient pas de développer dans ce chapitre car elles ne modifient pas le principe de fonctionnement de l'automate composé.

b) On peut simplifier un peu le travail du programmeur (et réduire la capacité de la mémoire de programme B) en diminuant la taille des nombres binaires codes des différentes instructions. Ici par exemple, nous n'avons que 9 instructions. 4 bits doivent être suffisants pour les codes. Si l'on n'est pas limité par la dimension de la mémoire de microprogramme A et en observant que chaque instruction occupe au maximum 7 lignes dans cette mémoire, on peut affecter à chaque instruction un espace de 8 lignes. Par exemple, l'instruction $R + 1$ occupe les sept lignes de l'adresse 0001000 à l'adresse 0001110, l'instruction $R - 1$ occupe les sept lignes

	L_A	CI_B	T	$(RD)_R$	$(WR)_R$	$(CI)_R$	$(RD)_D$	$(WR)_D$	$(CI)_D$	$(RD)_M$	$(WR)_M$	$(CI)_A$	$(CI)_Z$	$(RD)_Z$	L	C/\bar{D}	$(CI)_C$	$(RD)_C$
Chargement initial du compteur A	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
000010	1	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	0	0	1	1	1	0	1	1	0	1	1	0	0	1	1	0	0	0
	0	0	1	1	1	0	1	1	0	1	1	1	0	1	1	0	0	0
	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	1	0	0
R + 1	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	1	1	0
	0	0	1	1	1	1	1	1	0	1	1	0	0	1	0	0	0	0
	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	0
	1	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	0
001001	0	0	1	1	1	0	1	1	0	1	1	0	0	1	1	0	0	0
	0	0	1	1	1	0	1	1	0	1	1	1	0	1	1	0	0	0
	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	0
R - 1	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	1	0
	0	0	1	1	1	1	1	1	0	1	1	0	0	1	0	0	0	0
	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	0
	1	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	0
010000	0	0	1	0	1	0	1	0	0	1	1	0	0	1	0	0	0	1
	0	0	1	0	1	0	1	0	1	1	1	0	0	1	0	0	0	1
R → D	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	1	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
010100	0	0	1	1	0	0	1	0	1	0	1	0	0	1	0	0	0	1
	0	0	1	1	0	1	0	1	0	1	1	0	0	1	0	0	0	1
D → R	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	1	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
011000	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	0	0	1	1	1	0	1	1	0	1	1	1	0	1	0	0	0	1
M(R) → D	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	0	1
	0	0	1	1	1	0	1	0	1	0	1	0	0	1	0	0	0	1
	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	1	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
011110	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	0	0	1	1	1	0	1	1	0	1	1	1	0	1	0	0	0	1
D → M(R)	0	0	1	1	1	0	0	1	0	1	0	0	0	1	0	0	0	1
	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	1	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
100011	0	0	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	1
	0	0	1	1	1	0	1	1	0	1	1	1	0	0	0	0	0	1
	0	0	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0	1
D + M(R) → D	0	0	1	1	1	0	1	1	0	0	1	0	1	0	0	0	0	1
	0	0	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	1
	0	1	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	1
	1	0	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	1
101010	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	0	0	1	1	1	0	1	1	0	1	1	1	0	1	0	0	0	1
M(R) → R	0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	0	0	1
	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1
	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
	1	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
110000	0	0	0	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1
Arrêt	L_A	CI_B	T	$(RD)_R$	$(WR)_R$	$(CI)_R$	$(RD)_D$	$(WR)_D$	$(CI)_D$	$(RD)_M$	$(WR)_M$	$(CI)_A$	$(CI)_Z$	$(RD)_Z$	L	C/\bar{D}	$(CI)_C$	$(RD)_C$

Fig. 10

0 0	0	0	0	1	0	1	0	0	$D \rightarrow R$
0 1	0	0	0	0	0	0	1	0	$R + 1$
1 0	0	0	0	1	0	0	0	0	$R \rightarrow D$
1 1	0	0	1	1	0	0	0	0	Arrêt

Fig. 11: Macroprogrammation de $D + 1$

0 0 0	0	0	0	0	0	0	1	0	$R + 1$
0 0 1	0	0	0	1	1	0	0	0	$M(R) \rightarrow D$
0 1 0	0	0	0	0	0	0	1	0	$R + 1$
0 1 1	0	0	1	0	0	0	1	1	$D + M(R) \rightarrow D$
1 0 0	0	0	0	0	1	0	0	1	$R - 1$
1 0 1	0	0	0	0	1	0	0	1	$R - 1$
1 1 0	0	0	0	1	1	1	1	0	$D \rightarrow M(R)$
1 1 1	0	0	1	1	0	0	0	0	Arrêt

Fig. 12 : Macroprogrammation de l'opération $N_0 = N_1 + N_2$ avec stockage du résultat

de l'adresse 00 10 000 à l'adresse 0010 1 10, l'instruction $R \rightarrow D$ les quatre lignes de l'adresse 0011 000 à l'adresse 000110 11 etc... Les trois bits de plus faible poids de l'adresse de la première ligne de chacune de ces instructions sont toujours 000 : ils ne servent donc pas à reconnaître une instruction et il n'est pas nécessaire de les enregistrer en mémoire B. Seuls, les quatre bits de poids fort seront inscrits en mémoire de programme. Les trois bits de faible poids de l'entrée de chargement parallèle du compteur de microprogramme A seront maintenus à 0 de manière à bien charger, à la fin de chaque instruction, l'adresse complète du début de l'instruction suivante dans la mémoire A.

Chapitre 4

NOTION DE PROCESSEUR

I. Elaboration d'un processeur élémentaire

I. 1. Les paragraphes « assimilation » des deux chapitres précédents ont eu pour objet la réalisation de la commande d'un système organisé autour d'un même bus de données, à l'aide d'un automate programmable, simple ou composé. Considérons maintenant, d'une manière globale, *l'ensemble* formé par ce système à commander et l'automate programmable composé (fig. 1).

Cet ensemble comporte trois mémoires :

- la mémoire de microprogramme (mémoire A) adressée par le compteur de microprogramme (compteur A) qui fournit les microactions nécessaires à la commande de l'ensemble (système à commander proprement dit et automate lui-même).
- la mémoire de macroprogramme (mémoire B) adressée par le compteur de macroprogramme (compteur B). Cette mémoire contient la liste des *codes* des fonctions que l'on désire voir successivement se réaliser. Rappelons que chaque code est en fait l'adresse de la première ligne de l'espace de la mémoire A où est microprogrammée l'instruction correspondante.
- enfin la mémoire M, élément du système à commander, où, d'une part, sont rangées des données destinées à être fournies aux registres R et D ou à être « traitées » par l'additionneur Σ et où, d'autre part, on peut stocker d'autres données provenant du registre D. Cette mémoire est adressée par le registre R par l'intermédiaire du registre A.

Il importe d'avoir toujours présent à l'esprit la différence de nature des contenus de ces trois mémoires.

I. 2. Simplification de l'ensemble de la figure 1

Cette simplification consiste à regrouper les deux mémoires M et B en une seule

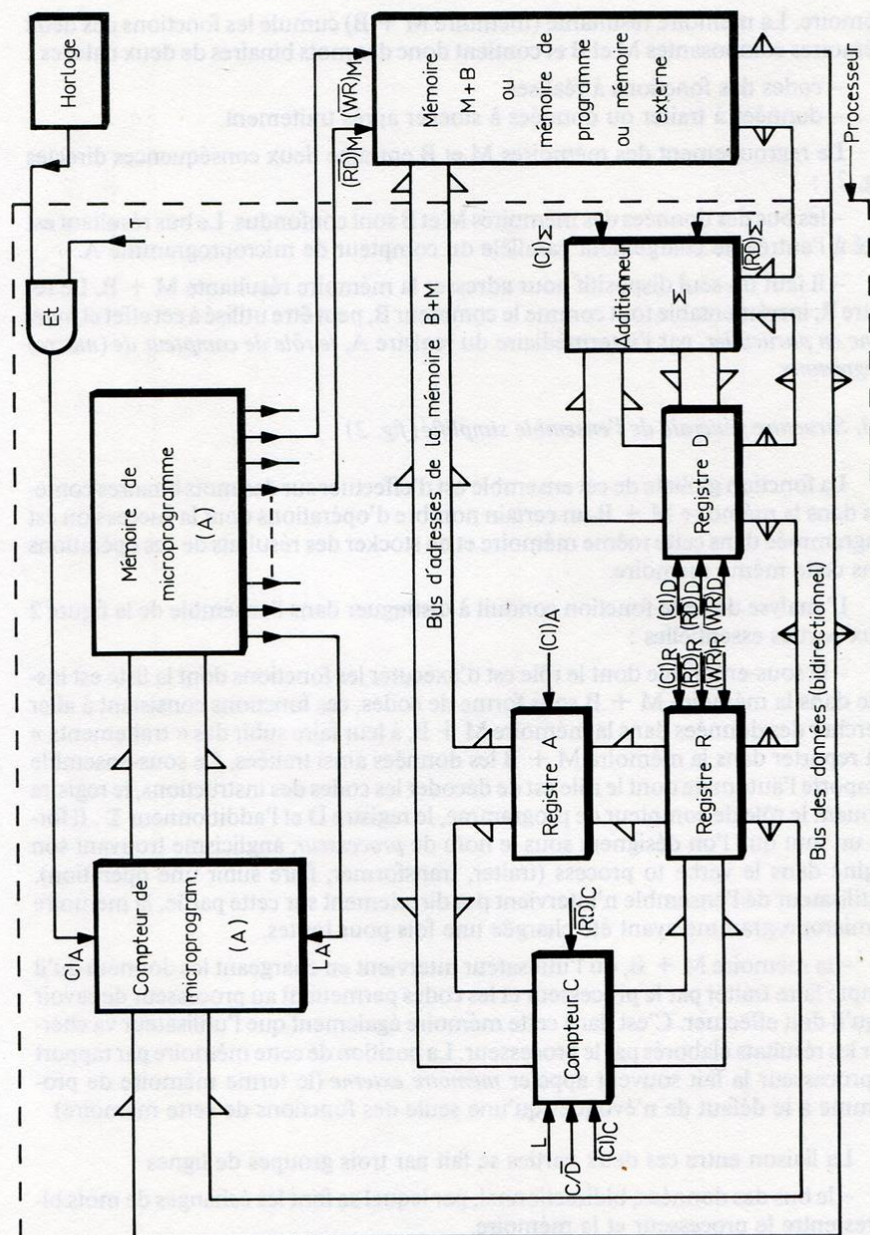


Fig. 2

mémoire. La mémoire résultante (mémoire $M + B$) cumule les fonctions des deux mémoires composantes M et B et contient donc des mots binaires de deux natures :

- codes des fonctions à réaliser
- données à traiter ou données à stocker après traitement.

Le regroupement des mémoires M et B entraîne deux conséquences directes (fig. 2) :

- les bus des données des mémoires M et B sont confondus. Le bus résultant est relié à l'entrée de chargement parallèle du compteur de microprogramme A .
- il faut un seul dispositif pour adresser la mémoire résultante $M + B$. Le registre R , incrémentable tout comme le compteur B , peut être utilisé à cet effet et *jouer donc en particulier, par l'intermédiaire du registre A , le rôle de compteur de (macro) programme.*

1. 3. Structure générale de l'ensemble simplifié (fig. 2)

La fonction globale de cet ensemble est d'effectuer sur des mots binaires contenus dans la mémoire $M + B$ un certain nombre d'opérations dont la succession est programmée dans cette même mémoire et de stocker des résultats de ces opérations dans cette même mémoire.

L'analyse de cette fonction conduit à distinguer dans l'ensemble de la figure 2 deux parties essentielles :

- le sous-ensemble dont le rôle est d'exécuter les fonctions dont la liste est inscrite dans la mémoire $M + B$ sous forme de codes, ces fonctions consistant à aller chercher des données dans la mémoire $M + B$, à leur faire subir des « traitements » et à reporter dans la mémoire $M + B$ les données ainsi traitées. Ce sous-ensemble comporte l'automate dont le rôle est de décoder les codes des instructions, le registre R jouant le rôle de compteur de programme, le registre D et l'additionneur Σ . Il forme un tout que l'on désignera sous le nom de *processeur*, anglicisme trouvant son origine dans le verbe to process (traiter, transformer, faire subir une opération). L'utilisateur de l'ensemble n'intervient pas directement sur cette partie, la mémoire de microprogramme ayant été chargée une fois pour toutes.

- la mémoire $M + B$, où l'utilisateur intervient en chargeant les données qu'il compte faire traiter par le processeur et les codes permettant au processeur de savoir ce qu'il doit effectuer. C'est dans cette mémoire également que l'utilisateur va chercher les résultats élaborés par le processeur. La position de cette mémoire par rapport au processeur la fait souvent appeler *mémoire externe* (le terme mémoire de programme a le défaut de n'évoquer qu'une seule des fonctions de cette mémoire).

La liaison entre ces deux parties se fait par trois groupes de lignes :

- le bus des données, bidirectionnel, par lequel se font les échanges de mots binaires entre le processeur et la mémoire.

- le bus des adresses (sous entendu : de la mémoire externe) qui fournit une adresse à la mémoire $M + B$.
- les lignes de contrôle, ici réduites aux supports des signaux $(\overline{RD})_M$ et $(\overline{WR})_M$.

II. Programmation du processeur élémentaire

II - 1. Microprogrammation de la phase de recherche de chaque instruction.

On se souvient que le microprogramme de chaque fonction élémentaire comporte deux parties : microprogramme d'exécution de l'instruction elle-même et microprogramme de recherche de l'instruction suivante.

S'il n'y a pas lieu de modifier le microprogramme d'exécution de l'instruction elle-même, il est nécessaire de reprendre le microprogramme de recherche puisque c'est maintenant le registre R qui fait office de compteur de programme et que la mémoire contenant les codes des instructions (mémoire externe) n'est pas toujours en fonction « lecture » comme l'était la mémoire B.

a) 1ère solution

La conception de la phase de recherche peut s'inspirer directement du fonctionnement de l'automate composé :

- après la fin de la phase d'exécution de l'instruction dont le code est inscrit à l'adresse $N - 1$ de la mémoire externe (pour simplifier l'exposé, appelons cette instruction : instruction d'adresse $N - 1$), on incrémente le registre R d'une unité dont le contenu passe donc à N. Cette opération est facilitée par le fait que, à la fin de la recherche de l'instruction précédente (instruction d'adresse $N - 1$), le registre A a été chargé par $N - 1$.

Pour incrémenter le registre R, il suffit donc d'incrémenter le compteur C puis de transférer le résultat N dans le registre R.

- Le contenu N du registre R est alors chargé dans le registre A et dans le compteur C (afin de préparer la recherche suivante).
- La mémoire externe alors correctement adressée est placée en fonction lecture : le bus des données supporte donc le code de l'instruction d'adresse N.
- Ce code est chargé dans le compteur de l'automate A.

La figure 3 représente la partie « recherche de l'instruction suivante » du microprogramme de chaque instruction. Comme ni le registre D, ni l'additionneur Σ n'in-

terviennent dans cette phase et que le registre R n'est pas relié au bus des données, on maintient $(\overline{RD})_D$, $(\overline{WR})_D$, $(\overline{RD})_\Sigma$, $(\overline{RD})_R$ et $(\overline{WR})_R$ au niveau logique 1 et $(Cl)_D$ et $(Cl)_\Sigma$ au niveau logique 0.

b). 2ème solution.

Le fait que le compteur de programme (ici le registre R) soit séparé de la mémoire de programme qu'il adresse par le registre A permet une autre mise en œuvre de la phase de recherche de l'instruction suivante. Faisons l'hypothèse que l'incréméntation du registre R ait été effectuée lors de la phase de recherche inscrite dans le microprogramme de l'instruction d'adresse N - 2, c'est-à-dire que pendant l'exécution de l'instruction d'adresse N - 1, le registre R contienne déjà le nombre N. La phase de recherche de l'instruction d'adresse N, microprogrammée à la fin de l'instruction d'adresse N - 1, comporte alors les étapes suivantes (fig. 4) :

- chargement du registre A par N : la mémoire externe est en mesure de fournir le code de l'instruction d'adresse N. Le compteur C contient déjà N et il ne semble pas nécessaire de la charger à nouveau par ce nombre N. Cependant, si l'instruction d'adresse N - 1 est un branchement (voir plus loin), ce n'est plus le cas et il est nécessaire de charger le compteur C par le nouveau contenu du registre R, par l'intermédiaire du registre A.
- incréméntation du compteur C puis transfert de son contenu N + 1 dans le registre R afin de préparer la phase de recherche de l'instruction d'adresse N + 1 suivant la phase d'exécution de l'instruction d'adresse N. Le contenu du registre A demeure N.
- positionnement de la mémoire externe en fonction lecture : le bus des données supporte alors le code de l'instruction d'adresse N.
- chargement de ce code dans le compteur de l'automate A.

Retenons bien que, dans ce procédé de mise en œuvre de la recherche de l'instruction suivante, pendant l'exécution de l'instruction d'adresse N, le compteur de programme (registre R) contient déjà le nombre N + 1.

Nous adopterons cette deuxième solution pour la suite de cet exposé : elle facilite la réalisation de certaines instructions. De plus, c'est aussi la solution adoptée dans le microprocesseur COSMAC CDP 1802.

II - 2. Microprogrammation d'instructions nouvelles induites par cette structure.

Il s'agit d'instructions mettant à profit l'ambivalence de la mémoire externe contenant à la fois des codes d'instructions et des données à traiter.

a) *Chargement immédiat du registre D.*

Supposons que le code de cette opération soit inscrit à l'adresse N de la mémoire externe. Elle consiste à charger dans le registre D le mot binaire inscrit dans la ligne d'adresse N + 1. L'instruction suivante a son code inscrit à l'adresse N + 2 (fig. 5).

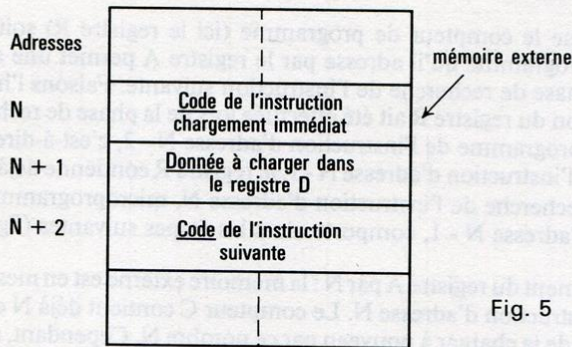


Fig. 5

On se souvient que, au début de la phase d'exécution de cette instruction d'adresse N, le registre R contient N + 1 du fait de la solution adoptée pour la phase de recherche. Cette phase d'exécution comporte :

- le chargement du registre A par le contenu N + 1 du registre R : la ligne d'adresse N + 1, contenant la donnée à charger dans le registre D, est pointée.
- l'établissement des liaisons entre la sortie de la mémoire et l'entrée du registre D : la donnée est présente à l'entrée du registre D.
- le chargement du registre D.

Cette séquence est celle décrite au paragraphe III - 1. b) du chapitre 2. C'est l'opération notée $M(R) \rightarrow D$.

La phase de recherche qui suit doit conduire à la reconnaissance de l'instruction suivante dont le code est inscrit non pas à l'adresse N + 1 occupée par la donnée mais à l'adresse N + 2. Cette phase de recherche suppose que le registre R contient déjà N + 2. Avant d'entreprendre cette phase de recherche, il est donc nécessaire d'incrémenter à nouveau le registre R, opération décrite au paragraphe III - 1. e) du chapitre 2 et notée $R + 1$.

La fonction chargement immédiat du registre D apparaît comme résultant de l'association dans cet ordre des fonctions $M(R) \rightarrow D$ et $R + 1$. Elle sera naturellement notée : $M(R) \rightarrow D, R + 1$. La figure 6 indique le microprogramme de cette instruction, en insistant sur sa composition.

	L_A	$(\overline{RD})_R$	$(\overline{WR})_R$	$(CI)_R$	$(\overline{RD})_D$	$(\overline{WR})_D$	$(CI)_D$	$(\overline{RD})_M$	$(\overline{WR})_M$	$(CI)_A$	$(CI)_\Sigma$	$(\overline{RD})_\Sigma$	L	C/\overline{D}	$(CI)_C$	$(\overline{RD})_C$
Code	0	1	1	0	1	1	0	1	1	1	0	1	0	0	0	1
	0	1	1	0	1	0	0	0	1	0	0	1	0	0	0	1
	0	1	1	0	1	0	1	0	1	0	0	1	0	0	0	1
	0	1	1	0	1	1	0	1	1	1	0	1	1	0	0	0
	0	1	1	0	1	1	0	1	1	0	0	1	0	1	0	0
	0	1	1	0	1	1	0	1	1	0	0	1	0	1	1	0
	0	1	1	1	1	1	0	1	1	0	0	1	0	0	0	0
	0	1	1	0	1	1	0	1	1	1	0	1	1	0	0	0
	0	1	1	0	1	1	0	1	1	0	0	1	0	1	0	0
	0	1	1	0	1	1	0	1	1	0	0	1	0	1	1	0
	0	1	1	1	1	1	0	0	1	0	0	1	0	0	0	0
	1	1	1	0	1	1	0	0	1	0	0	1	0	0	0	0

$M(R) \rightarrow D$

$R + 1$

Recherche instruction suivante

Fig. 6 :

Microprogramme complet de l'instruction chargement immédiat

Remarques

– Parmi la succession des mots binaires contenus dans la mémoire externe, le processeur est donc capable de différencier ceux représentant un *code* de ceux représentant une *donnée*. Dans l'exemple traité ici, c'est précisément le code de l'instruction chargement immédiat qui contient l'information permettant au mot binaire suivant d'être interprété comme une donnée par le processeur et non comme un ordre.

– Au cours de l'exécution de l'instruction chargement immédiat dont le code est inscrit à l'adresse N de la mémoire externe, le processeur doit charger dans le registre D la donnée stockée à l'adresse N + 1. A cet effet, il fournit à la mémoire externe l'adresse suivant *immédiatement* N. On dit qu'il y a un *adressage immédiat*.

b) addition immédiate

Cette opération, dont le code est inscrit à l'adresse N de la mémoire externe, consiste à additionner le contenu du registre D et le contenu de la ligne N + 1 de la mémoire externe et de charger le résultat dans le registre D (fig. 7).



Fig. 8

82

Comme dans le cas du chargement immédiat, cette instruction comporte deux étapes : d'abord $D + M(R) \rightarrow D$ puisque le contenu du compteur de programme (registre R) est $N + 1$ au début de la phase d'exécution, puis $R + 1$ de manière à préparer la phase de recherche de l'instruction suivante dont le code est inscrit à l'adresse $N + 2$ de la même mémoire externe. Cette instruction peut être notée symboliquement $D + M(R) \rightarrow D, R + 1$. Son microprogramme est reporté sur la figure 8.

c) *stockage immédiat.*

Il s'agit de transférer le contenu du registre D, dans la ligne d'adresse $N + 1$ de la mémoire suivant la ligne d'adresse N où est placé le code de cette instruction. Le lecteur comprendra sans peine qu'elle correspond à l'association dans cet ordre des fonctions $D \rightarrow M(R)$ et $R + 1$. Le code de l'instruction suivante est inscrit à l'adresse $N + 2$.

Remarque

Dans les processeurs industriels, le stockage immédiat n'existe pas. Il est remplacé par des instructions utilisant un autre registre pour fournir l'adresse à laquelle la donnée doit être stockée en mémoire. La solution de stockage adoptée ici (stockage immédiat) présente le seul intérêt de ne pas nécessiter une augmentation de la complexité de notre processeur élémentaire.

d) *branchement inconditionnel*

C'est une instruction de type particulier qui intervient sur le déroulement même du programme enregistré en mémoire externe. Soit N l'adresse de la ligne où est enregistré le code de cette instruction. La ligne d'adresse $N + 1$ contient elle-même l'adresse S où est inscrit le code de l'instruction qui doit être exécutée immédiatement après (fig. 9).

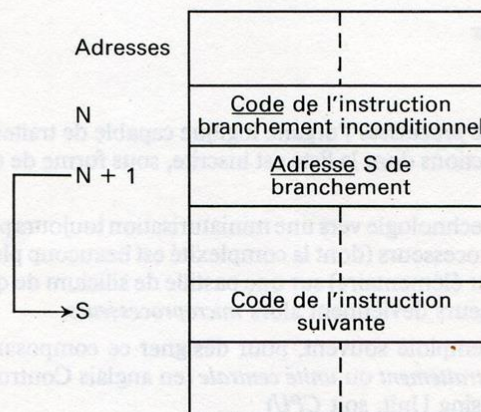


Fig. 9

Cette instruction de branchement permet des ruptures de séquence autorisant la réalisation de sauts de programme ou de boucles de programme dont on verra plus loin l'intérêt. Sa réalisation consiste uniquement à charger S dans le compteur de programme : c'est la fonction $M(R) \rightarrow R$. La phase de recherche qui suit son exécution amène S sur le registre A ; l'instruction d'adresse S sera donc bien traitée dans la prochaine phase d'exécution. Pendant cette phase de recherche S est chargé dans le compteur C (on comprend ici la nécessité de prévoir le transfert du registre A vers le compteur C), le compteur C est incrémenté et le résultat $S + 1$ de cette incrémentation est chargé dans le registre R préparant ainsi le prochain adressage.

Une nouvelle séquence peut commencer à partir de l'adresse S.

e) *Remarque.*

Pour les trois instructions chargement immédiat, addition immédiate et branchement inconditionnel, le code de l'opération est suivi, dans la mémoire externe, par un nombre binaire qui peut être soit une *donnée* à traiter, soit une *adresse* à partir de laquelle doit se continuer le programme. Dans les deux cas, le mot binaire inscrit immédiatement après le code de l'instruction est appelé *opérande*.

II - 3. Macroprogrammation

Le microprogramme de toutes les instructions étant figé dans une mémoire ROM, l'utilisateur du processeur a juste à effectuer le chargement de la mémoire externe par des codes d'instructions et éventuellement des opérands : c'est la (macro) programmation.

A titre d'exemple, la figure 10 donne le programme permettant au processeur d'exécuter la mission suivante : effectuer sur deux nombres binaires N_1 et N_2 , l'opération $N_1 + 2N_2$ et stocker le résultat à l'adresse S de la mémoire externe. On a mis en évidence les deux types de mots contenus dans la mémoire (codes et opérands).

III. Le microprocesseur

III - 1. Dénominations

Nous avons appelé *processeur* l'organe logique capable de traiter des données en exécutant des instructions dont la liste est inscrite, sous forme de codes, dans la mémoire externe.

L'évolution de la technologie vers une miniaturisation toujours plus poussée, a permis d'intégrer des processeurs (dont la complexité est beaucoup plus grande que celle de notre processeur élémentaire) sur une pastille de silicium de quelques mm^2 de surface. Ces processeurs deviennent alors *microprocesseurs*.

Notons que l'on emploie souvent, pour désigner ce composant, les termes *unité de contrôle et de traitement* ou *unité centrale* (en anglais Control and Process Unit ou Central Processing Unit, soit *CPU*)

Adresses		Commentaire
0	<u>Code</u> instruction $M(R) \rightarrow D, R+1$	
1	<u>Donnée</u> N_1	Registre D contient N_1
2	<u>Code</u> instruction $D+M(R) \rightarrow D, R+1$	
3	<u>Donnée</u> N_2	Registre D contient N_1+N_2
4	<u>Code</u> instruction $D+M(R) \rightarrow D, R+1$	
5	<u>Donnée</u> N_2	Registre D contient N_1+2N_2
6	<u>Code</u> instruction $M(R) \rightarrow R$	
7	<u>Adresse</u> de branchement $S-1$	
8		
S-1	<u>Code</u> instruction $D \rightarrow M(R), R+1$	
S		Ligne où sera stocké le résultat
S+1	<u>Code</u> instruction Arrêt	

Fig. 10

III - 2. Organes constitutifs fondamentaux d'un microprocesseur.

Etant donné la diversité des microprocesseurs existants, il est difficile de donner une organisation générale type. Notons cependant que tous comportent (fig. 11) :

a) un *compteur de programme*, dispositif destiné à fournir les adresses successives où sont enregistrés dans la mémoire externe les codes des instructions à exécuter. Le registre R de notre processeur élémentaire joue ce rôle.

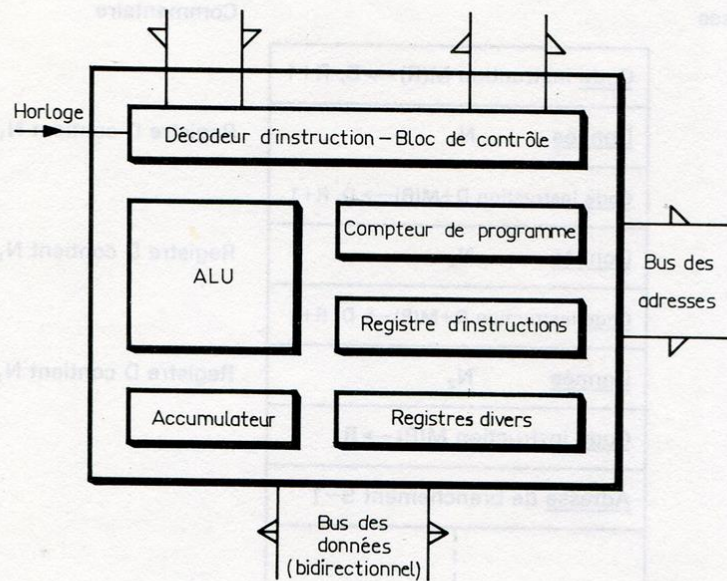


Fig. 11

b) *un registre d'instruction*. Lors de la phase de recherche d'une instruction, ce registre reçoit le code en provenance de la mémoire, via le bus des données. Cette fonction est remplie, dans le processeur élémentaire, par le compteur A lui-même.

c) *un décodeur d'instruction et bloc de contrôle*. Il interprète les codes contenus dans le registre d'instruction en tenant compte éventuellement de l'état de diverses entrées de contrôle. C'est lui qui fournit aux autres organes du microprocesseur toutes les microactions permettant l'exécution des instructions et c'est lui qui délivre au monde extérieur divers signaux permettant entre autres de contrôler la mémoire externe et les circuits de couplage aux périphériques. C'est l'automate A qui assure ces fonctions dans notre processeur élémentaire.

d) *une unité arithmétique et logique (ALU)*. C'est le centre de calcul du processeur. Elle est chargée d'opérations logiques ou arithmétiques sur des données en provenance généralement de la mémoire. Elle remplace – en agrandissant le champ des possibilités – l'additionneur Σ du processeur élémentaire.

e) *un (au moins) registre accumulateur*. C'est sur son contenu – entre autres – que l'ALU effectue des opérations. Il reçoit les résultats de ces opérations. C'est souvent un intermédiaire entre l'ALU et le monde extérieur au microprocesseur.

f) *des registres de travail divers*.

III - 3. Environnement du microprocesseur.

Nous savons déjà que le (micro) processeur ne peut pas travailler seul. Il est indispensable qu'il soit associé à une mémoire. La liaison avec la mémoire externe se fait par les bus des adresses, des données et de contrôle. Notons au passage que la mémoire externe peut être matériellement réalisée par l'association de plusieurs boîtiers mémoire, en particulier de types différents (RAM et ROM). En effet, les codes des fonctions à réaliser et certaines des données à traiter peuvent être avantageusement figés en ROM lors d'une application industrielle ; dans ce cas, l'ensemble mémoire externe contient une RAM destinée à stocker des données après traitement.

Il est bien évident que le système microprocesseur - mémoire ne présente un intérêt que s'il est associé à un ou des organes *périphériques*. Si la mission de l'ensemble microprocesseur - mémoire est d'effectuer la *commande* d'un dispositif, ce dispositif constitue un périphérique. Dans le cas où le microprocesseur effectue du *traitement* de données, un périphérique au moins apporte au système les données à traiter ; après traitement, ces données sont redistribuées à d'autres périphériques. La liaison entre le système microprocesseur - mémoire et les organes périphériques se fait par l'intermédiaire de circuits dits d'interfaçage ou de couplage (fig. 12). Nous n'en dirons pas plus, chaque famille de microprocesseurs ayant une technique particulière de réalisation de cette liaison.

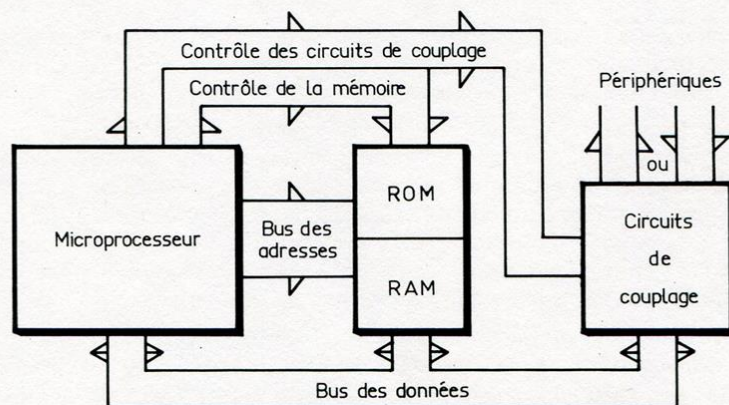


Fig. 12

Chapitre 5

STRUCTURE DU MICROPROCESSEUR COSMAC CDP 1802

Une bonne utilisation du microprocesseur COSMAC suppose une bonne connaissance de son organisation interne.

La figure 1 représente le schéma synoptique du microprocesseur COSMAC. Rappelons que les flèches placées sur les bus indiquent le sens dans lequel peuvent transiter des mots binaires sur ces bus. On a placé entre parenthèses le nombre de bits des différents registres et le nombre de conducteurs des différents bus.

Un examen rapide de ce schéma synoptique permet les conclusions suivantes :

- on retrouve, organisés autour d'un même bus de données, certains éléments rencontrés dans le processeur élémentaire que nous avons élaboré au chapitre précédent (registre (s) R, registre A, organe d'incrément/décément, additionneur Σ inclus dans l'ALU).

- la complexité du système est augmentée par l'adjonction de nouveaux registres (N, P, X, T, I, DF, IE), la multiplicité des registres R, la mise en œuvre d'un multiplexeur, d'une bascule Q, le remplacement de l'additionneur Σ par une unité logique et arithmétique (ALU). La conséquence en est que les possibilités de ce microprocesseur sont *beaucoup* plus grandes que celles de notre processeur élémentaire.

Nous nous proposons d'entreprendre une description *fonctionnelle* des différents organes du microprocesseur.

I. L'ensemble décodeur d'instruction et bloc de contrôle

Cet ensemble joue le rôle de l'automate A de notre processeur élémentaire. Il reçoit, en provenance de la mémoire externe et par l'intermédiaire du bus des données et des deux registres I et N les *codes* des instructions à exécuter. Il fournit séquentiellement à tout le reste du système les microactions nécessaires pour l'exécu-

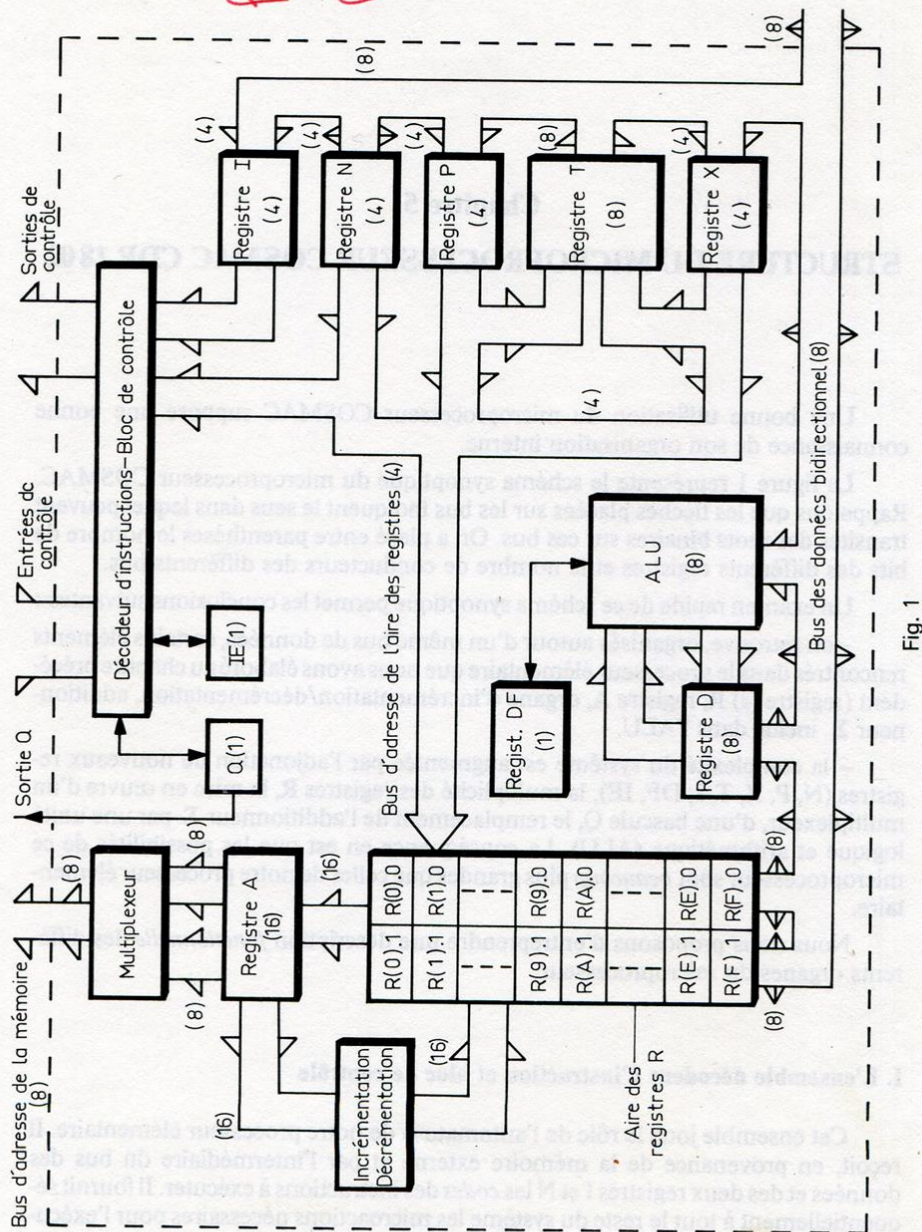


Fig. 1

tion de ces instructions (signaux d'horloge des divers registres, commandes des amplificateurs trois états...). Il reçoit et interprète également les ordres provenant d'entrées de contrôle et fournit au monde extérieur un certain nombre de signaux de contrôle (comme par exemple les signaux MRD et MWR de la mémoire externe).

II. Le bus des données

Il comporte 8 lignes et permet donc de véhiculer des mots binaires de 8 bits que l'on appelle des *octets* (bytes en anglais).

Le rôle du bus des données est, comme dans notre processeur élémentaire, d'assurer la liaison entre différents organes du processeur lui-même (registres R, ALU, registre D) mais aussi d'assurer la liaison entre le processeur, la mémoire externe et éventuellement les périphériques, par l'intermédiaire des organes d'entrée/sortie. Ce bus amène ainsi au processeur les codes des instructions puisés dans la mémoire ; il véhicule les données à traiter par le processeur inscrites dans la mémoire et les données traitées par le processeur à inscrire en mémoire ; il transporte les octets à échanger entre la mémoire et les organes d'entrée/sortie.

La taille de ce bus range le COSMAC dans la catégorie des *microprocesseurs 8 bits*.

III. Les registres R et les éléments associés

Notre processeur élémentaire ne comportait qu'un seul registre R à qui nous avons attribué le rôle bien particulier de compteur de programme. Dans les chapitres 2 et 3, nous avons utilisé ce même registre pour pointer une ligne de la mémoire B.

Dans le COSMAC, les registres R sont au nombre de 16, notés respectivement registres R(0), R(1),..., R(E), R(F) (utilisation de la numération hexadécimale). Ces registres ont chacun une capacité de 16 bits.

Comme on a besoin d'effectuer des échanges entre ces registres et le registre D et la mémoire externe par l'intermédiaire du bus des données constitué lui-même de 8 fils, on est obligé de séparer chaque registre R(W) en deux parties de 8 bits chacune, R(W).1, R(W).0, R(W).1 supportant l'octet de poids fort et R(W).0 l'octet de poids faible. L'entrée ou la sortie de chacun des 32 demi-registres de 8 bits ainsi obtenus peut être reliée au bus des données grâce à l'intervention d'amplificateurs trois états.

Le contenu d'un registre R(W) (16 bits) peut être chargé dans un registre A et adresser une mémoire externe dont la capacité peut aller jusqu'à 64 Koctets (rappelons que, conventionnellement, un Koctet correspond à 1 024 octets, soit 64 Koctets

= 65 536 octets = 2^{16} octets). Le bus des adresses étant limité à 8 fils, il est nécessaire de procéder à un multiplexage, le bus des adresses supportant d'abord l'octet de poids fort puis l'octet de poids faible de l'adresse de 16 bits, sous contrôle du processeur lui-même (voir plus loin).

Le contenu du registre A peut également être chargé dans un dispositif d'incrément/décément restituant au registre R(W) son contenu augmenté ou diminué d'une unité.

Le contenu d'un registre R peut donc être dirigé, soit vers le bus des données (en deux fois), soit vers le bus des adresses (après multiplexage), soit vers le dispositif d'incrément/décément.

IV. Les différents rôles des registres R. Les registres pointeurs P, X, N.

Les registres R sont au nombre de 16. Un registre parmi les 16 peut donc être sélectionné par un mot de 4 bits, contenu dans l'un des registres P, X, N. Il s'agit en quelque sorte d'un adressage de l'aire des registres R par le registre P, X ou N de 4 bits.

IV. 1 Compteur de programme

L'un des registres R est utilisé comme compteur de programme (comme nous l'avons fait avec notre processeur élémentaire). Le registre P désigne le registre R à qui est attribué ce rôle qui, rappelons-le, consiste à fournir les adresses successives de la mémoire externe où est enregistré le programme à exécuter.

On notera P le contenu du registre P, R(P) le contenu du registre R(P) pointé par le registre P et M(R(P)) l'octet inscrit dans la mémoire externe à l'adresse R(P) fournie par le registre R(P) désigné par le registre P (fig. 2).

Le choix du registre R qui joue le rôle de compteur de programme est laissé à l'entière discrétion de l'utilisateur qui fixe le contenu P du registre P par programme. On notera cependant que :

- une procédure d'initialisation (voir plus loin) a en particulier pour conséquence de mettre à zéro le registre P et le registre R(0). Juste après initialisation, le compteur de programme est alors le registre R(0) qui pointe l'adresse 0 de la mémoire externe puisque son contenu est 0.

- lorsqu'une opération DMA (direct acces memory = accès direct à la mémoire externe) a lieu, le registre R(0) est alors également automatiquement compteur de programme.

- lors du traitement d'une interruption (voir plus loin), $P = 1$ et le registre R(1) est obligatoirement compteur de programme.

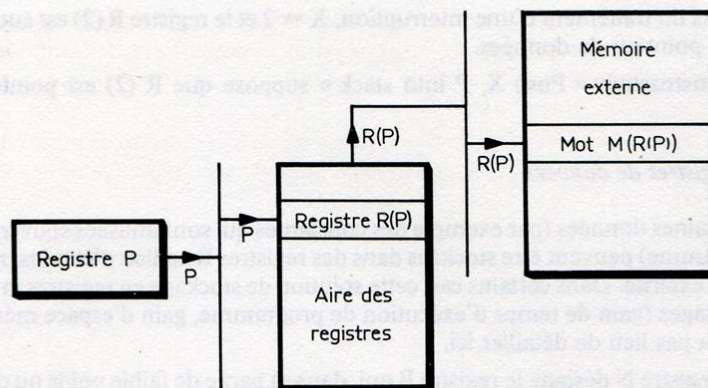


Fig. 2

Comme nous le verrons, la possibilité qui est offerte à l'utilisateur de choisir celui qui, parmi les 16 registres R, est le compteur de programme facilite l'introduction des sous-programmes.

IV. 2. Pointeur de données

Généralement, une partie (de type RAM) de la mémoire externe est utilisée pour recevoir, au cours du déroulement d'un programme, des résultats intermédiaires qui peuvent être réutilisés dans la suite du programme. Ces résultats intermédiaires sont souvent stockés dans des lignes de mémoire adjacentes et forment une *pile* de données (stack) du type LIFO (last in, first out : dernier entré, premier sorti).

L'un des registres R est utilisé pour adresser cet espace mémoire réservé aux données (espace des données). Le registre X désigne le registre R à qui est attribué ce rôle qui consiste à fournir l'adresse de la mémoire externe où doit être, soit stocké un résultat intermédiaire ou une donnée à sauvegarder, soit lue une donnée.

On notera X le *contenu* du registre X, R(X) le *contenu* du registre R(X) pointé par le registre X et M(R(X)) l'octet inscrit (ou qui va être inscrit) dans la mémoire externe à l'adresse R(X) fournie par le registre R(X) désigné par le registre X.

Le choix du registre R qui joue le rôle de pointeur de données est également laissé à la discrétion de l'utilisateur qui peut fixer le contenu X du registre X par programme. Cependant, il faut savoir que :

- à l'issue d'une procédure d'initialisation, $X = 0$. Comme $P = 0$ aussi, R(0) est compteur de programme et il convient de choisir un autre registre R comme pointeur de pile de données.

– lors du traitement d'une interruption, $X = 2$ et le registre R (2) est automatiquement pointeur de données.

– l'instruction « Push X, P into stack » suppose que R (2) est pointeur de données.

IV. 3. Registres de données

Certaines données (par exemple des constantes qui sont utilisées souvent dans un programme) peuvent être stockées dans des registres R au lieu d'être inscrites en mémoire externe. Dans certains cas, cette solution de stockage en registres présente des avantages (gain de temps d'exécution de programme, gain d'espace mémoire) qu'il n'y a pas lieu de détailler, ici.

Le registre N désigne le registre R qui, dans sa partie de faible poids ou dans sa partie de fort poids, doit, soit recevoir une donnée provenant du registre D, soit fournir une donnée au registre D. Le registre N pointe également le registre R qui doit être incrémenté ou décrémenté. Il peut encore désigner le registre fournissant l'adresse d'une ligne de la mémoire externe.

Bien entendu, on notera R (N) le *contenu* du registre R (N) pointé par le registre N de *contenu* N.

V. L'ALU, le registre D et le registre D F

L'additionneur Σ du processeur élémentaire est, dans le microprocesseur COSMAC, remplacé par une unité arithmétique et logique (ALU) qui permet d'effectuer diverses opérations logiques (ou, ou exclusif, et) et arithmétiques (additions et soustractions avec ou sans retenues). Les deux opérandes intervenant dans chacune de ces opérations sont fournis à l'ALU, directement par le registre D pour l'un, par la mémoire externe via le bus des données pour l'autre. L'octet résultant est toujours chargé dans le registre D que l'on appelle *accumulateur*. Dans le cas d'opérations arithmétiques, le registre monobit D F reçoit la retenue éventuelle. Cette retenue peut être prise en compte lors d'une opération identique faite sur deux octets de poids plus grand, ce qui permet par exemple l'addition et la soustraction de nombres binaires de plus de 8 bits.

L'ALU exécute aussi les opérations de décalages à gauche (ou à droite) du contenu du registre D. Le bit initial de poids le plus fort (ou le plus faible) peut être sauvegardé en le chargeant dans le registre D F.

Le registre D peut être chargé, en vue d'une opération à effectuer par l'ALU, par le contenu d'une ligne de la mémoire externe ou d'un demi-registre R et réciproquement le résultat d'une opération qu'il reçoit de l'ALU peut être transféré dans un demi-registre R ou une case mémoire.

Signalons enfin que le contenu 0 ou 1 du registre D F (data flag) et la nullité du contenu du registre D peuvent être testés dans les certaines instructions (branchements et sauts).

VI. Le registre T

C'est un registre de 8 bits destiné à recevoir *temporairement* le contenu des registres X et P. Son contenu peut être stocké dans une pile de données pour être éventuellement restitué aux registres X et P.

VII. Les registres d'instruction I et N

Lors de la phase de recherche d'une instruction, l'octet, code de cette instruction, est lu dans la mémoire externe et, par l'intermédiaire du bus des données, les 4 bits de poids fort sont chargés dans le registre I et les 4 bits de poids faible dans le registre N.

Dans la phase d'exécution, le décodeur d'instruction interprète les contenus de ces deux registres d'instructions.

Souvent, le registre I indique le type de l'instruction. Le registre N peut indiquer suivant le type des instructions :

- le registre R sur lequel une opération doit être effectuée (rôle de pointeur déjà évoqué).
- la nature d'une opération que l'ALU doit effectuer.
- la valeur qui doit être chargée dans les registres P ou X lors de l'exécution de l'instruction appropriée.
- l'organe d'entrée/sortie sélectionné pour une opération d'entrée/sortie.

VIII. Les bascules Q et I E

La capacité de ces bascules est de un bit.

L'état logique (0 ou 1) de la bascule Q est programmable et testable dans certaines instructions (branchements, sauts). Elle constitue une *sortie* du microprocesseur, très précieuse du fait de la simplicité de sa mise en œuvre (pas d'intervention d'organe d'entrée/sortie).

L'état 1 de la bascule I E (interrupt enable) autorise la prise en compte d'une demande d'interruption (voir plus loin). Lorsqu'une demande d'interruption est acceptée par le microprocesseur, la bascule IE passe à l'état 0 ce qui interdit la prise en compte d'une autre demande d'interruption tant que celle qui est en cours n'est pas terminée.

Notons que, à l'issue d'une procédure d'initialisation, on a $Q = 0$ et $IE = 1$.

Chapitre 6

LES INSTRUCTIONS DU MICROPROCESSEUR COSMAC CDP 1802

I. Considérations générales

Le microprocesseur COSMAC possède un jeu de 91 instructions.

On se souvient que la réalisation d'une instruction dont le code est inscrit dans la mémoire externe se fait toujours en deux étapes :

- *recherche* de l'instruction : le microprocesseur charge dans les registres I et N l'octet code de l'instruction à réaliser et le compteur de programme s'incrémente d'une unité.

- *exécution* de l'instruction : le décodeur reconnaît l'instruction et fournit séquentiellement toutes les microactions nécessaires à sa réalisation.

I. 1. Durée de réalisation d'une instruction

La phase de recherche d'une instruction nécessite toujours huit périodes d'horloge.

Le décodeur est conçu de manière que la phase d'exécution de la *plupart* des instructions requiert également huit périodes d'horloge.

Cet intervalle de temps correspondant à huit périodes d'horloge est une caractéristique du microprocesseur que l'on appelle *cycle machine*. La réalisation d'une instruction se fait donc généralement en *deux cycles machine* (soit seize périodes d'horloge), un cycle étant nécessaire à la recherche de l'instruction (c'est le *cycle de recherche*) et un autre à son exécution (c'est le *cycle d'exécution*).

Cependant, l'exécution de certaines instructions (branchements longs, saut long, pas d'opération) nécessite des microactions en nombre trop grand pour pouvoir être effectuée en huit périodes d'horloge. L'exécution est alors étalée sur seize périodes d'horloge soit deux cycles machine. Pour réaliser de telles instructions, il faut donc *trois cycles machine* (soit vingt quatre périodes d'horloge), un cycle de recherche et deux cycles d'exécution.

binaire	hexadécimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Tableau 1

I. 2. Format d'une instruction (fig. 1)

Le format d'une instruction est le nombre de lignes de la mémoire externe que cette instruction occupe, c'est-à-dire le nombre d'octets qu'elle nécessite

a) instructions à un octet

Ce sont toutes les instructions qui ne nécessitent pas d'opérande. L'octet unique est le *code* de l'instruction à effectuer.

b) instructions à deux octets ou trois octets

Certaines instructions font intervenir un ou des opérandes. C'est le cas de toutes les opérations « immédiates » où la ligne de la mémoire externe qui suit celle où est inscrit le code de l'instruction contient une *donnée* à charger dans le registre D ou dans l'ALU. C'est le cas aussi des opérations de branchement où, après le code de l'opération, il faut inscrire l'*adresse* de branchement. Le compteur de programme R(P) dont il faut modifier le contenu pour effectuer le branchement est un registre de 16 bits. Si la modification n'affecte que sa partie de faible poids (8 bits) un seul octet

supplémentaire sera inscrit en mémoire : il s'agit d'un branchement court. Si la modification affecte la partie de fort poids et la partie de faible poids, il faut inscrire en mémoire, sous le code de l'opération, les deux octets de la nouvelle adresse : il s'agit d'un branchement long.

Les instructions opérations immédiates et branchements courts nécessitent donc deux octets (un pour le code, un pour l'opérande donnée ou adresse), les branchements longs nécessitent trois octets.

Remarque : Les deux paragraphes précédents sont liés à des considérations pratiques importantes : durée d'exécution d'un programme complet par le microprocesseur, occupation optimale de la mémoire externe dans le cas d'un programme conséquent.

I. 3 Mnémonique et code d'une instruction

Chaque instruction possède une désignation. Par exemple, l'instruction qui conduit à l'opération $M(R(P)) \rightarrow D$ puis $R(P) + 1$ s'appelle « chargement immédiat » (sous entendu « du registre D ») ou en anglais « load immediate ». A la désignation (anglaise) de cette instruction est associée une abréviation mnémotechnique comportant trois ou quatre lettres que l'on appelle la *mnémonique* de l'instruction. Ainsi, la mnémonique de l'instruction « load immediate » est LDI.

Chaque instruction possède un *code* qui peut être considéré comme la traduction binaire de la mnémonique. Le code de l'instruction LDI est l'octet 11111000. Lors de la recherche de cette instruction, les quatre bits de fort poids 1111 sont chargés dans le registre I, les quatre bits de faible poids 1000 dans le registre N. Il n'est pas très commode d'écrire de tels nombres binaires de huit chiffres. On préfère leur substituer leur équivalent hexadécimal à deux chiffres. Ainsi le code de l'instruction LDI s'écrit, en hexadécimal, F8. Le tableau 1 rappelle les résultats de la conversion binaire – hexadécimal. Pour trouver la représentation hexadécimale d'un mot binaire de 8 bits, il faut considérer que celui-ci résulte de l'association des 4 bits de poids fort et des 4 bits de poids faible ; à chacun de ces demi-octets correspond un caractère hexadécimal.

L'association de ces deux caractères constitue la représentation hexadécimale du mot binaire de 8 bits. Par exemple, l'octet (01111011) a pour représentation hexadécimale 7B parce qu'au demi-octet (0111) est associé 7 et au demi-octet (1011) est associé B.

II. Jeu d'instructions du microprocesseur COSMAC

Il est donné au tableau 2.

Les troisième et quatrième parties de cet ouvrage commentent et utilisent la plupart de ces instructions. Nous nous dispenserons donc ici de toute explication.

Tableau 2 : jeu d'instructions 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	$R(N).0 \rightarrow D$
PUT LOW REG N	PLO	AN	1	2	$D \rightarrow R(N).0$
GET HIGH REG N	GHI	9N	1	2	$R(N).1 \rightarrow D$
PUT HIGH REG N	PHI	BN	1	2	$D \rightarrow R(N).1$
Opérations de transfert entre l'accumulateur D et la mémoire.					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
LOAD VIA N	LDN	0N	1	2	$M(R(N)) \rightarrow D$; FOR N NOT 0
LOAD ADVANCE	LDA	4N	1	2	$M(R(N)) \rightarrow D$; $R(N) + 1$
LOAD VIA X	LDX	F0	1	2	$M(R(X)) \rightarrow D$
LOAD VIA X AND ADVANCE	LDXA	72	1	2	$M(R(X)) \rightarrow D$; $R(X) + 1$
LOAD IMMEDIATE	LDI	F8	2	2	$M(R(P)) \rightarrow D$; $R(P) + 1$
STORE VIA N	STR	5N	1	2	$D \rightarrow M(R(N))$
STORE VIA X AND DECREMENT	STXD	73	1	2	$D \rightarrow M(R(X))$; $R(X) - 1$

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 2 : jeu d'instructions du CDP 1802 (suite)

Opérations logiques.					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
OR	OR	F1	1	2	$M(R(X)) \text{ OR } D \longrightarrow D$
OR IMMEDIATE	ORI	F9	2	2	$M(R(P)) \text{ OR } D \longrightarrow D ; R(P) + 1$
EXCLUSIVE OR	XOR	F3	1	2	$M(R(X)) \text{ XOR } D \longrightarrow D$
EXCLUSIVE OR IMMEDIATE	XRI	FB	2	2	$M(R(P)) \text{ XOR } D \longrightarrow D ; R(P) + 1$
AND	AND	F2	1	2	$M(R(X)) \text{ AND } D \longrightarrow D$
AND IMMEDIATE	ANI	FA	2	2	$M(R(P)) \text{ AND } D \longrightarrow D ; R(P) + 1$
SHIFT RIGHT	SHR	F6	1	2	SHIFT D RIGHT, LSB (D) \longrightarrow DF, 0 \longrightarrow MSB (D)
SHIFT RIGHT WITH CARRY	SHRC	76	1	2	SHIFT D RIGHT, LSB (\tilde{D}) \longrightarrow DF, DF \longrightarrow MSB (D)
RING SHIFT RIGHT	RSHR				
SHIFT LEFT	SHL	FE	1	2	SHIFT D LEFT, MSB (D) \longrightarrow DF, 0 \longrightarrow LSB (D)
SHIFT LEFT WITH CARRY	SHLC	7E	1	2	SHIFT D LEFT, MSB (D) \longrightarrow DF, DF \longrightarrow LSB (D)
RING SHIFT LEFT	RSHL				
Opérations arithmétiques.					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
ADD	ADD	F4	1	2	$M(R(X)) + D \longrightarrow \text{DF}, D$
ADD IMMEDIATE	ADI	FC	2	2	$M(R(P)) + D \longrightarrow \text{DF}, D ; R(P) + 1$
ADD WITH CARRY	ADC	74	1	2	$M(R(X)) + D + \text{DF} \longrightarrow \text{DF}, D$
ADD WITH CARRY IMMEDIATE	ADCI	7C	2	2	$M(R(P)) + D + \text{DF} \longrightarrow \text{DF}, D ; R(P) + 1$
SUBSTRACT D	SD	F5	1	2	$M(R(X)) - D \longrightarrow \text{DF}, D$
SUBSTRACT D IMMEDIATE	SDI	FD	2	2	$M(R(P)) - D \longrightarrow \text{DF}, D ; R(P) + 1$
SUBSTRACT D WITH BORROW	SDB	75	1	2	$M(R(X)) - D - (\text{NOT DF}) \longrightarrow \text{DF}, D$
SUBSTRACT D WITH BORROW, IMMEDIATE	SDBI	7D	2	2	$M(R(P)) - D - (\text{NOT DF}) \longrightarrow \text{DF}, D ; R(P) + 1$
SUBSTRACT MEMORY	SM	F7	1	2	$D - M(R(X)) \longrightarrow \text{DF}, D$
SUBSTRACT MEMORY IMMEDIATE	SMI	FF	2	2	$D - M(R(P)) \longrightarrow \text{DF}, D ; R(P) + 1$
SUBSTRACT MEMORY WITH BORROW	SMB	77	1	2	$D - M(R(X)) - (\text{NOT DF}) \longrightarrow \text{DF}, D$
SUBSTRACT MEMORY WITH BORROW, IMMEDIATE	SMBI	7F	2	2	$D - M(R(P)) - (\text{NOT DF}) \longrightarrow \text{DF}, D ; R(P) + 1$

Tableau 2 : jeu d'instructions du CDP 1802 (suite)

Instructions de branchement. (Branchement court).					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
SHORT BRANCH	BR	30	2	2	$M(R(P)) \rightarrow R(P).0$
NO SHORT BRANCH (voir SKP)	NBR	38	2	2	$R(P) + 1$
SHORT BRANCH IF D=0	BZ	32	2	2	IF D=0, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF D NOT 0	BNZ	3A	2	2	IF D NOT 0, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF DF=1	BDF	33	2	2	IF DF=1, $M(R(P)) \rightarrow R(P).0$ 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, $M(R(P)) \rightarrow R(P).0$ 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, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF Q=0	BNQ	39	2	2	IF Q=0, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF EF1=1	BI	34	2	2	IF EF1=1, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF EF1=0	BNI	3C	2	2	IF EF1=0, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF EF2=1	B2	35	2	2	IF EF2=1, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF EF2=0	BN2	3D	2	2	IF EF2=0, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF EF3=1	B3	36	2	2	IF EF3=1, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF EF3=0	BN3	3E	2	2	IF EF3=0, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF EF4=1	B4	37	2	2	IF EF4=1, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$
SHORT BRANCH IF EF4=0	BN4	3F	2	2	IF EF4=0, $M(R(P)) \rightarrow R(P).0$ ELSE $R(P) + 1$

Tableau 2 : jeu d'instructions du CDP 1802 (suite)

Instructions de branchement. (Branchement long).					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
LONG BRANCH	LBR	C0	3	3	$M(R(P)) \rightarrow R(P).1$ $M(R(P)+1) \rightarrow R(P).0$ $R(P) + 2$
NO LONG BRANCH (voir LSKP)	NLBR	C8	3	3	
LONG BRANCH IF D=0	LBZ	C2	3	3	IF D=0, $M(R(P)) \rightarrow R(P).1$ $M(R(P)+1) \rightarrow R(P).0$ ELSE $R(P) + 2$
LONG BRANCH IF D NOT 0	LBNZ	CA	3	3	IF D NOT 0, $M(R(P)) \rightarrow R(P).1$ $M(R(P)+1) \rightarrow R(P).0$ ELSE $R(P) + 2$
LONG BRANCH IF DF=1	LBDF	C3	3	3	IF DF=1, $M(R(P)) \rightarrow R(P).1$ $M(R(P)+1) \rightarrow R(P).0$ ELSE $R(P) + 2$
LONG BRANCH IF DF=0	LBNF	CB	3	3	IF DF=0, $M(R(P)) \rightarrow R(P).1$ $M(R(P)+1) \rightarrow R(P).0$ ELSE $R(P) + 2$
LONG BRANCH IF Q=1	LBQ	C1	3	3	IF Q=1, $M(R(P)) \rightarrow R(P).1$ $M(R(P)+1) \rightarrow R(P).0$ ELSE $R(P) + 2$
LONG BRANCH IF Q=0	LBNQ	C9	3	3	IF Q=0, $M(R(P)) \rightarrow R(P).1$ $M(R(P)+1) \rightarrow R(P).0$ 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 2 : jeu d'instructions du CDP 1802 (fin)

Instructions de contrôle.					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
IDLE	IDL	00	1	2	WAIT FOR DMA OR INTERRUPT ; $M(R(0)) \rightarrow \text{BUS}$
NO OPÉRATION	NOP	C4	1	3	CONTINUE
SET P	SEP	DN	1	2	$N \rightarrow P$
SET X	SEX	EN	1	2	$N \rightarrow X$
SET Q	SEQ	7B	1	2	$1 \rightarrow Q$
RESET Q	REQ	7A	1	2	$0 \rightarrow Q$
SAVE	SAV	78	1	2	$T \rightarrow M(R(X))$
PUSH X,P TO STACK	MARK	79	1	2	$(X,P) \rightarrow T ; (X,P) \rightarrow M(R(2))$ THEN $P \rightarrow X ; R(2) - 1$
RETURN	RET	70	1	2	$M(R(X)) \rightarrow (X,P) ; R(X) + 1$ $1 \rightarrow IE$
DISABLE	DIS	71	1	2	$M(R(X)) \rightarrow (X,P) ; R(X) + 1$ $0 \rightarrow IE$
Instructions d'entrée/sortie					
INSTRUCTION	MNÉMONIQUE	CODE	OCTETS	CYCLES	OPÉRATION
OUTPUT 1	OUT 1	61	1	2	$M(R(X)) \rightarrow \text{BUS} ; R(X) + 1$ $N \text{ LINES} = 1$
OUTPUT 2	OUT 2	62	1	2	$M(R(X)) \rightarrow \text{BUS} ; R(X) + 1$ $N \text{ LINES} = 2$
OUTPUT 3	OUT 3	63	1	2	$M(R(X)) \rightarrow \text{BUS} ; R(X) + 1$ $N \text{ LINES} = 3$
OUTPUT 4	OUT 4	64	1	2	$M(R(X)) \rightarrow \text{BUS} ; R(X) + 1$ $N \text{ LINES} = 4$
OUTPUT 5	OUT 5	65	1	2	$M(R(X)) \rightarrow \text{BUS} ; R(X) + 1$ $N \text{ LINES} = 5$
OUTPUT 6	OUT 6	66	1	2	$M(R(X)) \rightarrow \text{BUS} ; R(X) + 1$ $N \text{ LINES} = 6$
OUTPUT 7	OUT 7	67	1	2	$M(R(X)) \rightarrow \text{BUS} ; R(X) + 1$ $N \text{ LINES} = 7$
INPUT 1	INP 1	69	1	2	$\text{BUS} \rightarrow M(R(X)) ; \text{BUS} \rightarrow D$ $N \text{ LINES} = 1$
INPUT 2	INP 2	6A	1	2	$\text{BUS} \rightarrow M(R(X)) ; \text{BUS} \rightarrow D$ $N \text{ LINES} = 2$
INPUT 3	INP 3	6B	1	2	$\text{BUS} \rightarrow M(R(X)) ; \text{BUS} \rightarrow D$ $N \text{ LINES} = 3$
INPUT 4	INP 4	6C	1	2	$\text{BUS} \rightarrow M(R(X)) ; \text{BUS} \rightarrow D$ $N \text{ LINES} = 4$
INPUT 5	INP 5	6D	1	2	$\text{BUS} \rightarrow M(R(X)) ; \text{BUS} \rightarrow D$ $N \text{ LINES} = 5$
INPUT 6	INP 6	6E	1	2	$\text{BUS} \rightarrow M(R(X)) ; \text{BUS} \rightarrow D$ $N \text{ LINES} = 6$
INPUT 7	INP 7	6F	1	2	$\text{BUS} \rightarrow M(R(X)) ; \text{BUS} \rightarrow D$ $N \text{ LINES} = 7$

Deuxième partie

CONCEPTION ET RÉALISATION D'UNE MAQUETTE D'AIDE A L'ÉTUDE DE LA MISE EN ŒUVRE ET DE LA PROGRAMMATION D'UN MICROPROCESSEUR

Introduction

Le but de cette partie est de démontrer que la seule lecture attentive des caractéristiques du constructeur d'un μP (1) permet à un électronicien n'ayant aucune connaissance en informatique d'utiliser efficacement ce μP à l'intérieur d'une application donnée.

Cependant, l'utilisation correcte d'un μP est subordonnée à son étude intime, étude que nous proposons de développer à l'aide d'une maquette que nous appellerons dorénavant maquette A. Nous conseillons au lecteur de réaliser la maquette A après avoir lu attentivement les lignes concernant sa conception.

L'intérêt qu'il y a à réaliser cette maquette réside dans le fait que le lecteur dispose alors d'un support matériel propre à lui permettre d'accéder à la compréhension immédiate du rôle de chaque broche ou ensemble de broches du μP . Par ailleurs, cette maquette constitue une aide précieuse lors de la mise au point d'un programme correspondant à une application à développer.

Pour l'essentiel, la maquette A est constituée du μP CDP 1802 (RCA) couplé à une mémoire RAM CDP 1823 (RCA) de capacité 128 octets ; le brochage de ces deux composants est donné figure 1.

Les raisons du choix de ce μP sont d'ordre purement pédagogique et résident principalement dans les observations suivantes :

1. Tous les registres internes au μP étant statiques, la fréquence minimale d'horloge requise est égale à 0. Autrement dit, le μP peut fonctionner en pas-à-pas. Il est prévu sur la maquette A un bouton poussoir, associé à une logique anti-rebond, dont la fonction est de générer des fronts montants sur l'entrée horloge (CL) du μP à un rythme laissé à l'entière discrétion de l'utilisateur ; celui-ci dispose par consé-

(1) μP veut dire microprocesseur et évite les répétitions fastidieuses de ce mot.

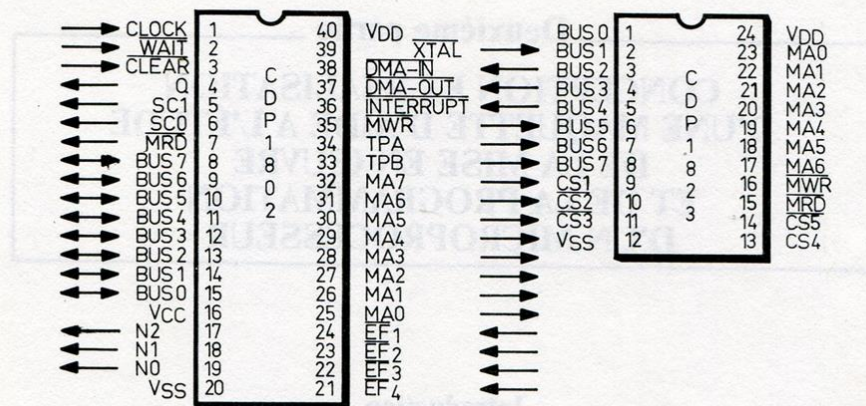


Fig. 1

quent de tout le temps nécessaire à l'étude de la coordination des signaux fournis ou reçus par le μP , ainsi que de la faculté de faire dérouler un programme en pas-à-pas tout en contrôlant les diverses procédures d'échanges de mots binaires entre le μP , la mémoire, et le monde extérieur.

2. Le μP CDP 1802 est évidemment en mesure de lire et d'exécuter une liste d'instructions écrites dans la mémoire ; il présente en outre la propriété remarquable d'être microprogrammé pour permettre d'entrer directement un programme dans la mémoire, en utilisant seulement une bascule comme composant additionnel (voir plus loin le mode « LOAD », c'est-à-dire le mode « Chargement » de la mémoire).

3. Le μP CDP 1802 présente une sortie Q programmable et quatre entrées notées EF_1 , EF_2 , EF_3 , EF_4 dites « drapeaux » (FLAG en anglais) dont l'état logique peut être testé par programme. La sortie Q et les entrées « drapeaux » réalisent ainsi une liaison directe avec le monde extérieur ; par conséquent il est possible de développer des applications ne nécessitant pas l'utilisation de circuits d'interface entre le système « μP - Mémoire » et le monde extérieur. Autrement dit, l'utilisateur peut rapidement être confronté à la mise en œuvre d'un système simple à μP .

Plutôt que de faire une description exhaustive du rôle des différentes broches du μP , et donc de présenter telles quelles les caractéristiques fournies par le constructeur, nous allons nous attacher à résoudre les problèmes fondamentaux que rencontre un électronicien désireux d'insérer un μP dans un système automatique. C'est dans cet esprit que nous développerons la conception de la maquette A, et le rôle des différentes broches du μP apparaîtra au fur et à mesure de notre étude.

Cependant, il convient de dire que la maquette A est le fruit d'une recherche non seulement théorique mais expérimentale, et par conséquent il est conseillé de la réaliser et de l'utiliser même si tous les problèmes relatifs aux caractéristiques du μP

ne sont pas clairement élucidés. En effet, c'est précisément l'un des rôles de la maquette de permettre d'élucider progressivement ces problèmes grâce à sa faculté de fonctionnement en pas-à-pas. Il est donc recommandé de réaliser rapidement la maquette A et d'adopter une procédure de travail qui consiste à effectuer un fréquent va-et-vient entre la partie de l'ouvrage consacrée à la conception de la maquette, et celle consacrée au développement des principes fondamentaux de la mise en œuvre et de la programmation des μP . Cette méthode présente l'avantage de permettre une accession rapide à la technique des μP , et assure à chaque instant la liaison entre les problèmes de logiciel (programmation) et les problèmes de matériel (interfaçage des composants).

Nous avons prévu une extension de la maquette A en concevant une maquette B destinée à lui être couplée, et qui présente une série d'améliorations dont le rôle est d'une part de permettre de développer des applications de plus grande envergure, d'autre part de faciliter la mise au point des programmes.

Chapitre 7

CONCEPTION D'UNE MAQUETTE D'AIDE A L'ÉTUDE DE LA MISE EN ŒUVRE ET DE LA PROGRAMMATION D'UN MICROPROCESSEUR

Dans ce chapitre, le microprocesseur est considéré comme un composant électronique. L'étude des caractéristiques du constructeur est nécessaire et suffisante à l'élaboration de l'ensemble maquette A - maquette B. Nous proposons au lecteur de suivre le cheminement même qui a permis aux auteurs, électroniciens, de concevoir un système bâti autour du microprocesseur COSMAC CDP 1802. Les raisons de ce choix ont déjà été annoncées ; signalons cependant que la structure de l'étude que nous allons développer peut s'adapter à tout autre microprocesseur.

1. L'horloge

L'entrée CL du microprocesseur est destinée à recevoir des impulsions de fréquence comprise entre 0 et 3,2 MHz si le microprocesseur est alimenté sous 5 V (ce qui est le cas de notre maquette) ou comprise entre 0 et 6,4 MHz si le microprocesseur est alimenté sous 10 V.

Ces impulsions peuvent être générées selon deux modes distincts :

- le mode « pas-à-pas »
- le mode « astable ».

Un inverseur (fig. 1) permet la sélection de l'un ou l'autre de ces modes en reliant l'entrée CL du microprocesseur à :

- un circuit anti-rebonds associé à un bouton poussoir dans le mode « pas-à-pas »
- un oscillateur astable dans le mode « astable ».

Le circuit anti-rebonds utilise un demi boîtier NAND CD 4011 ; l'autre moitié sert à la fabrication d'un oscillateur de fréquence réglable par un circuit RC (fig. 2).

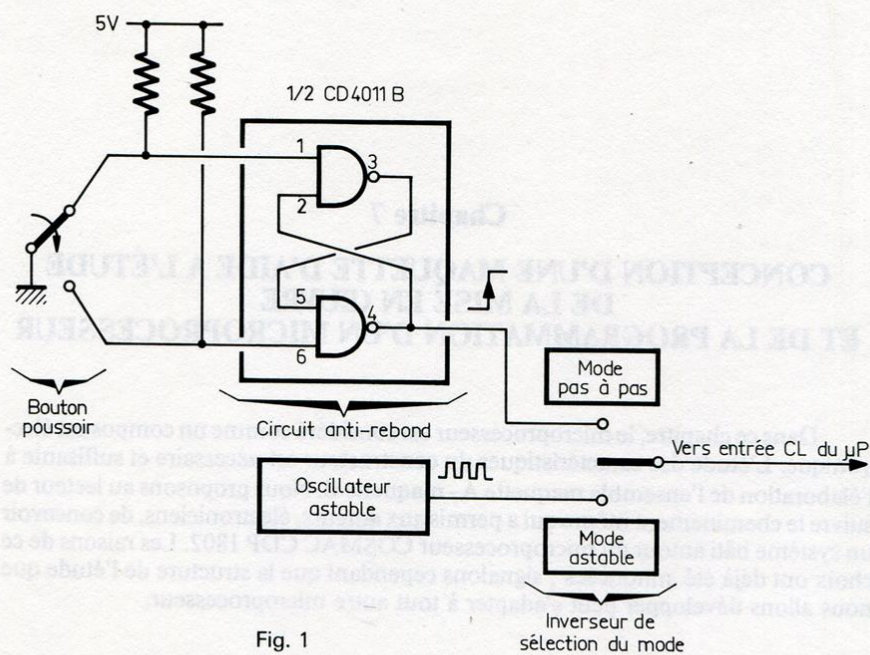


Fig. 1

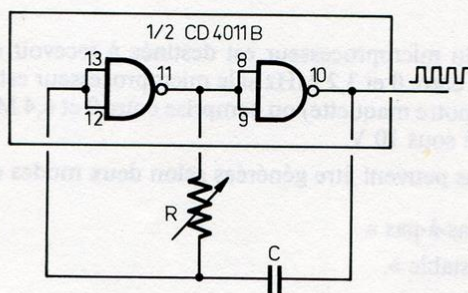


Fig. 2

La structure simplifiée de cet oscillateur résulte du seul fait que nous désirons utiliser le minimum de composants dans la fabrication de notre maquette ; cependant cet avantage nous oblige à considérer les défauts inhérents à cette structure et qui sont essentiellement les suivants :

1. Le temps de montée t_M du signal fourni par l'oscillateur dépend des valeurs numériques de R et de C, et risque dans certains cas d'être trop grand. En effet, le

constructeur du microprocesseur annonce que celui-ci se déclenche seulement si $t_M < 15 \mu s$. Cette difficulté ne peut être levée qu'expérimentalement en choisissant des valeurs correctes pour R et C. Pour notre part, et compte tenu des fréquences que nous désirons obtenir, nous avons décidé d'imposer $C = 47 \text{ nF}$ et R variable de $1,3 \text{ k}\Omega$ à quelques mégohms ; l'excursion de fréquence est alors de quelques hertz à environ 5 kilohertz.

2. la période T du signal fourni par l'oscillateur n'est pas calculable et est donnée par la relation $T = \alpha \cdot RC$ dans laquelle le coefficient α vaut approximativement 1,7 mais seulement aux très basses fréquences. Si ce n'est pas le cas, α est une fonction croissante de la fréquence dont l'expression mathématique n'est pas pratiquement exploitable. Par ailleurs, pour des valeurs de R et de C données, α dépend du temps et par suite la stabilité de cet oscillateur est médiocre.

L'étude des principes fondamentaux de la programmation et de la mise en œuvre d'un microprocesseur n'est aucunement troublée par la méconnaissance de la valeur exacte de T ni par sa dérive éventuelle. Cependant pour permettre à l'utilisateur s'il le désire, de développer des applications pour lesquelles des temporisations précises et stables dans le temps sont requises, nous avons prévu d'autoriser l'installation d'un quartz sur la maquette A. Dans ce but, une horloge est intégrée au microprocesseur et sa sortie notée XTAL est à relier à l'entrée CL par l'intermédiaire d'un circuit résonant constitué par un quartz associé à deux condensateurs d'environ 20 pF et une résistance de quelques mégohms (fig. 3).

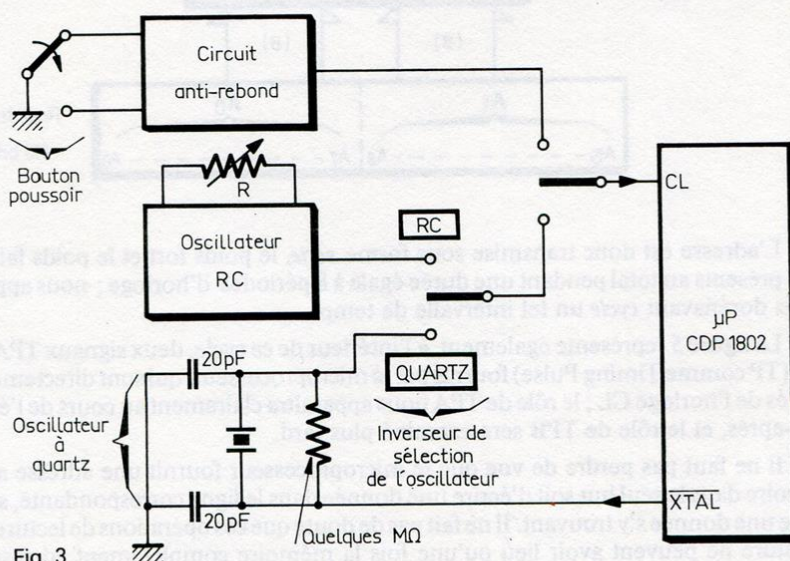


Fig. 3

II. Le couplage du microprocesseur à la mémoire

II - 1. Le bus d'adresses

Le compteur de programme P.C. interne au microprocesseur et dont nous reparlerons plus tard, présente une capacité de 16 bits et est donc susceptible d'adresser une mémoire de capacité maximale $2^{16} = 64$ kmots (1 kmots = 1024 mots).

On peut penser que la transmission d'une adresse de ce format vers la mémoire doit donc nécessiter le support d'un bus d'adresses constitué par 16 fils. Or il n'en est rien ; le bus d'adresses du COSMAC est constitué par 8 fils et permet cependant la transmission d'une adresse de format 16 bits. Comment ce résultat est-il obtenu ?

Supposons que le contenu de P.C. soit actuellement dans le registre noté A (fig. 4) et appelons les 8 bits de poids faible de l'adresse et les 8 bits de poids fort respectivement A_0 et A_1 ; si A_0 et A_1 sont placés sur les deux voies d'entrée d'un multiplexeur dont la sortie M A (comme Memory Address) constitue le bus d'adresses, il apparaît sur celui-ci *successivement* A_1 et A_0 et ceci à un rythme défini par les chronogrammes de la figure 5.

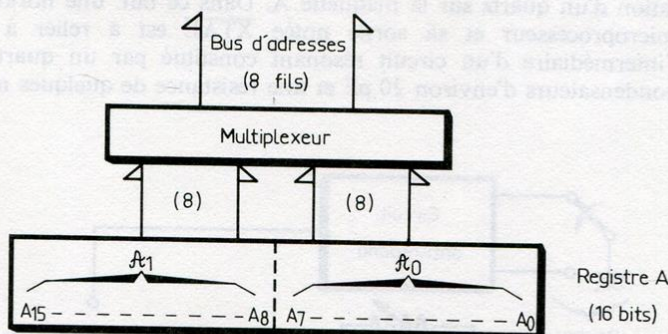


Fig. 4

L'adresse est donc transmise sous forme *série*, le poids fort et le poids faible étant présents au total pendant une durée égale à 8 périodes d'horloge ; nous appellerons dorénavant *cycle* un tel intervalle de temps.

La figure 5 représente également, à l'intérieur de ce cycle, deux signaux TPA et TPB (TP comme Timing Pulse) fournis par le microprocesseur, qui sont directement dérivés de l'horloge CL ; le rôle de TPA nous apparaîtra clairement au cours de l'étude ci-après, et le rôle de TPB sera examiné plus tard.

Il ne faut pas perdre de vue que le microprocesseur fournit une adresse à la mémoire dans le seul but soit d'écrire une donnée dans la ligne correspondante, soit de lire une donnée s'y trouvant. Il ne fait pas de doute que ces opérations de lecture et d'écriture ne peuvent avoir lieu qu'une fois la mémoire complètement adressée,

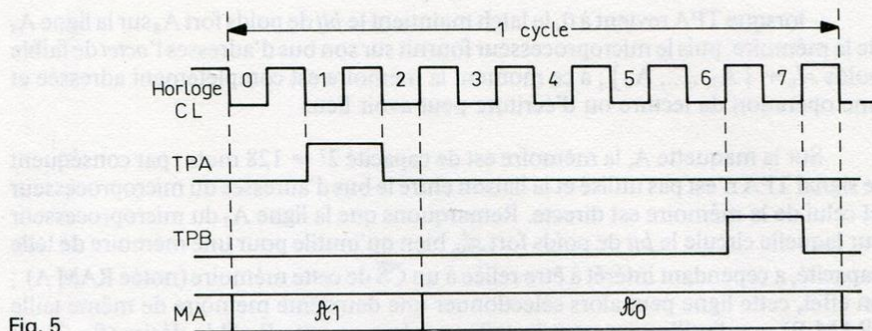


Fig. 5

c'est-à-dire exclusivement pendant que le poids faible \mathcal{A}_0 est présent sur le bus d'adresses et à condition que le poids fort \mathcal{A}_1 (ayant disparu au moment de l'apparition de \mathcal{A}_0) ait été mémorisé au cours de son apparition.

Cette condition peut être facilement réalisée en constatant que TPA est au niveau 1 seulement lorsque le poids fort \mathcal{A}_1 est présent ; en effet, TPA peut et doit alors servir à déclencher un registre latch destiné à mémoriser \mathcal{A}_1 . Par exemple (voir fig. 6), le bus d'adresses d'une mémoire de capacité $2^9 = 512$ mots est constitué par 9 fils dont celui relatif au *bit* de poids fort A_8 doit être relié à l'entrée d'un latch déclenché par TPA :

– si $TPA = 1$, le *bit* A_8 de poids fort de l'adresse qui appartient à \mathcal{A}_1 est présent sur la ligne A_0 et est transféré au travers le latch sur la ligne A_8 de la mémoire.

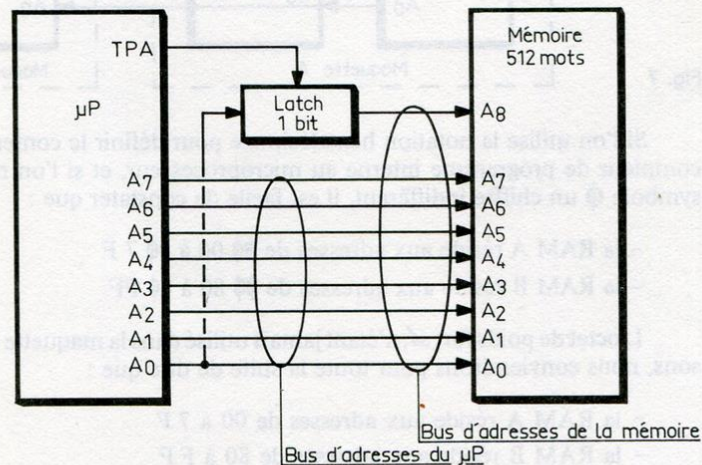


Fig. 6

– lorsque TPA revient à 0, le latch maintient le *bit* de poids fort A_8 sur la ligne A_8 de la mémoire, puis le microprocesseur fournit sur son bus d'adresses l'*octet* de faible poids $\mathcal{A}_0 = \{A_7, \dots, A_0\}$; à ce moment la mémoire est complètement adressée et une opération de lecture ou d'écriture peut avoir lieu.

Sur la maquette A, la mémoire est de capacité $2^7 = 128$ mots ; par conséquent le signal TPA n'est pas utilisé et la liaison entre le bus d'adresses du microprocesseur et celui de la mémoire est directe. Remarquons que la ligne A_7 du microprocesseur sur laquelle circule le *bit* de poids fort \mathcal{A}_0 , bien qu'inutile pour une mémoire de telle capacité, a cependant intérêt à être reliée à un \overline{CS} de cette mémoire (notée RAM A) ; en effet, cette ligne peut alors sélectionner une deuxième mémoire de même taille (RAM B) que l'utilisateur peut installer sur la maquette B s'il le désire (fig. 7). Si $A_7 = 0$, la RAM B n'est pas sélectionnée et est en haute impédance ; la RAM A, sélectionnée, est adressée par les bits A_6 à A_0 . Si $A_7 = 1$, c'est la RAM B qui est sélectionnée et qui est adressée par les bits de A_6 à A_0 . L'ensemble RAM A, RAM B constitue ainsi une mémoire de capacité 256 octets.

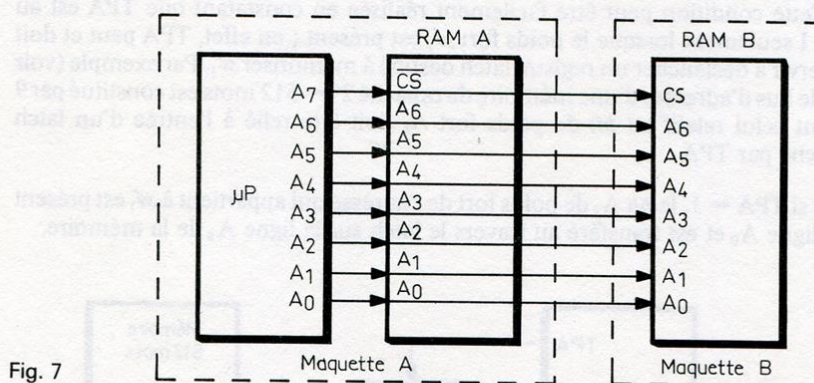


Fig. 7

Si l'on utilise la notation hexadécimale pour définir le contenu de 16 bits du compteur de programme interne au microprocesseur, et si l'on représente par le symbole Φ un chiffre indifférent, il est facile de constater que :

- la RAM A réside aux adresses de $\Phi\Phi 00$ à $\Phi\Phi 7 F$
- la RAM B réside aux adresses de $\Phi\Phi 80$ à $\Phi\Phi FF$

L'octet de poids fort \mathcal{A}_1 n'étant jamais utilisé dans la maquette que nous proposons, nous conviendrons pour toute la suite de dire que :

- la RAM A réside aux adresses de 00 à 7 F
- la RAM B réside aux adresses de 80 à F F

Pour que l'utilisateur soit en mesure de contrôler à tout instant l'adressage de la RAM, il est nécessaire d'afficher les bits A_7, \dots, A_0 présents sur le bus d'adresses. Des considérations d'ordre économique et de facilité du dessin d'un circuit imprimé simple face nous ont conduit à opter pour un affichage par diodes électroluminescentes (LED) pour la maquette A. Chaque ligne A_i du bus d'adresses est donc reliée, par l'intermédiaire d'une résistance R_B de $33\text{ k}\Omega$, à la base d'un transistor dont la mise en saturation commande l'allumage d'une LED ; le flux lumineux est suffisant si celle-ci est parcourue par un courant de l'ordre de 15 mA ajusté par la résistance R_C de $220\text{ }\Omega$ (fig. 8).

L'utilisateur peut cependant disposer d'un affichage alphanumérique de l'adresse en insérant des afficheurs hexadécimaux TIL 311 (Texas Instrument) à l'emplacement prévu à cet effet sur la maquette B. Une logique de décodage étant

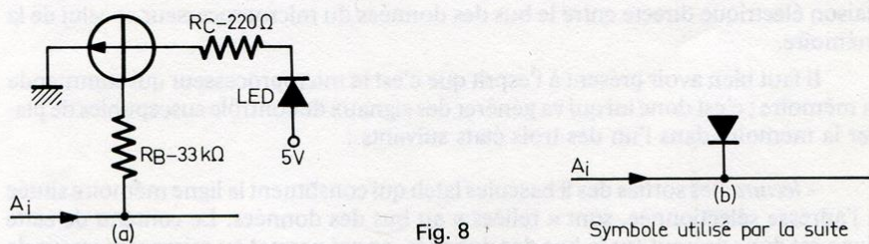


Fig. 8

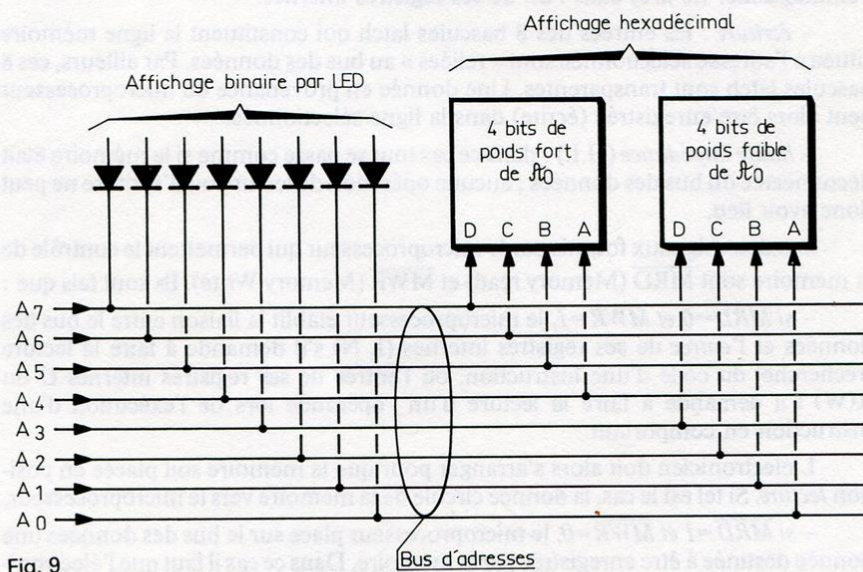


Fig. 9

intégrée à chacun de ces afficheurs, ceux-ci constituent une boîte noire à 4 entrées binaires { D, C, B, A } dont la sortie est l'affichage hexadécimal du mot binaire DCBA ; comme le microprocesseur est d'autre part compatible avec la technologie TTL adoptée pour ces afficheurs, ils peuvent être directement reliés au bus d'adresses (fig. 9).

II - 2. Le contrôle des échanges de données entre le microprocesseur et la mémoire

Au cours du fonctionnement d'un système à microprocesseur, celui-ci échange constamment des données avec la mémoire ; ces données circulent du microprocesseur vers la mémoire ou inversement en utilisant le bus des données comme voie de transmission. Le bus des données, qui est donc bidirectionnel, est constitué par la liaison électrique directe entre le bus des données du microprocesseur et celui de la mémoire.

Il faut bien avoir présent à l'esprit que c'est le microprocesseur qui commande la mémoire ; c'est donc lui qui va générer des signaux de contrôle susceptibles de placer la mémoire dans l'un des trois états suivants :

- *lecture* : les sorties des 8 bascules latch qui constituent la ligne mémoire située à l'adresse sélectionnée, sont « reliées » au bus des données. Le contenu de cette ligne est donc présent sur le bus des données, ce qui permet au microprocesseur de l'emmagasiner (le lire) dans l'un de ses registres internes.

- *écriture* : les entrées des 8 bascules latch qui constituent la ligne mémoire située à l'adresse sélectionnée sont « reliées » au bus des données. Par ailleurs, ces 8 bascules latch sont transparentes. Une donnée en provenance du microprocesseur peut alors être enregistrée (écrite) dans la ligne sélectionnée.

- *haute impédance (H.I.)* : dans ce cas tout se passe comme si la mémoire était déconnectée du bus des données ; aucune opération de lecture ou d'écriture ne peut donc avoir lieu.

Les deux signaux fournis par le microprocesseur qui permettent le contrôle de la mémoire sont \overline{MRD} (Memory read) et \overline{MWR} (Memory Write). Ils sont tels que :

- si $\overline{MRD}=0$ et $\overline{MWR}=1$, le microprocesseur établit la liaison entre le bus des données et l'entrée de ses registres internes (I, N) s'il demande à faire la lecture (recherche) du code d'une instruction, ou l'entrée de ses registres internes D ou R(W) s'il demande à faire la lecture d'un opérande lors de l'exécution d'une instruction en comportant.

L'électronicien doit alors s'arranger pour que la mémoire soit placée en position *lecture*. Si tel est le cas, la donnée circule de la mémoire vers le microprocesseur.

- si $\overline{MRD}=1$ et $\overline{MWR}=0$, le microprocesseur place sur le bus des données une donnée destinée à être enregistrée par la mémoire. Dans ce cas il faut que l'électroni-

cien s'arrange pour que la mémoire soit placée dans la position *écriture*. La donnée circule alors du microprocesseur vers la mémoire.

- si $\overline{MRD} = 1$ et $\overline{MWR} = 1$, le microprocesseur ne demande pas à échanger de données avec la mémoire, et celle-ci doit donc alors être placée dans l'état H.I.

- le microprocesseur ne fournit jamais la combinaison $\overline{MRD} = 0$ et $\overline{MWR} = 0$

L'analyse que nous venons de faire nous montre que les signaux \overline{MRD} et \overline{MWR} sont des requêtes de mise en position lecture, écriture ou H.I. de la mémoire. Par conséquent il est nécessaire de câbler celle-ci de telle sorte que ces requêtes puissent être satisfaites. Prenons deux exemples :

1. Supposons que la mémoire dont nous disposons soit contrôlable par les deux signaux R/\overline{W} et CE répondant à la définition suivante

- si $CE = 1$ et $R/\overline{W} = 1$, la mémoire est en position lecture.
- si $CE = 1$ et $R/\overline{W} = 0$, la mémoire est en position écriture.
- si $CE = 0$ et quel que soit R/\overline{W} , la mémoire est en H.I.

Il apparaît que :

- CE doit être au niveau 0 si et seulement si $\overline{MRD} = 1$ et $\overline{MWR} = 1$, et par conséquent $CE = \overline{MRD} \cdot \overline{MWR}$
- R/\overline{W} peut être directement relié à \overline{MWR} .

Le microprocesseur contrôlera donc correctement une mémoire de ce type en utilisant une porte NAND pour la génération du signal CE (voir fig. 10)

2. Nous pouvons éviter l'adjonction d'un boîtier NAND en utilisant une mémoire contrôlable par deux signaux notés également \overline{MRD} et \overline{MWR} et dont la définition répond exactement aux requêtes établies par le microprocesseur. La

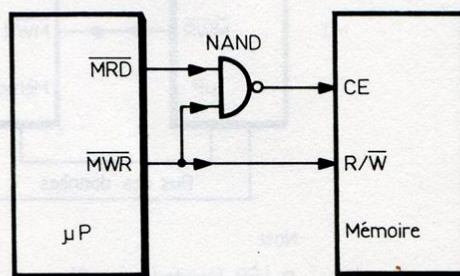


Fig. 10

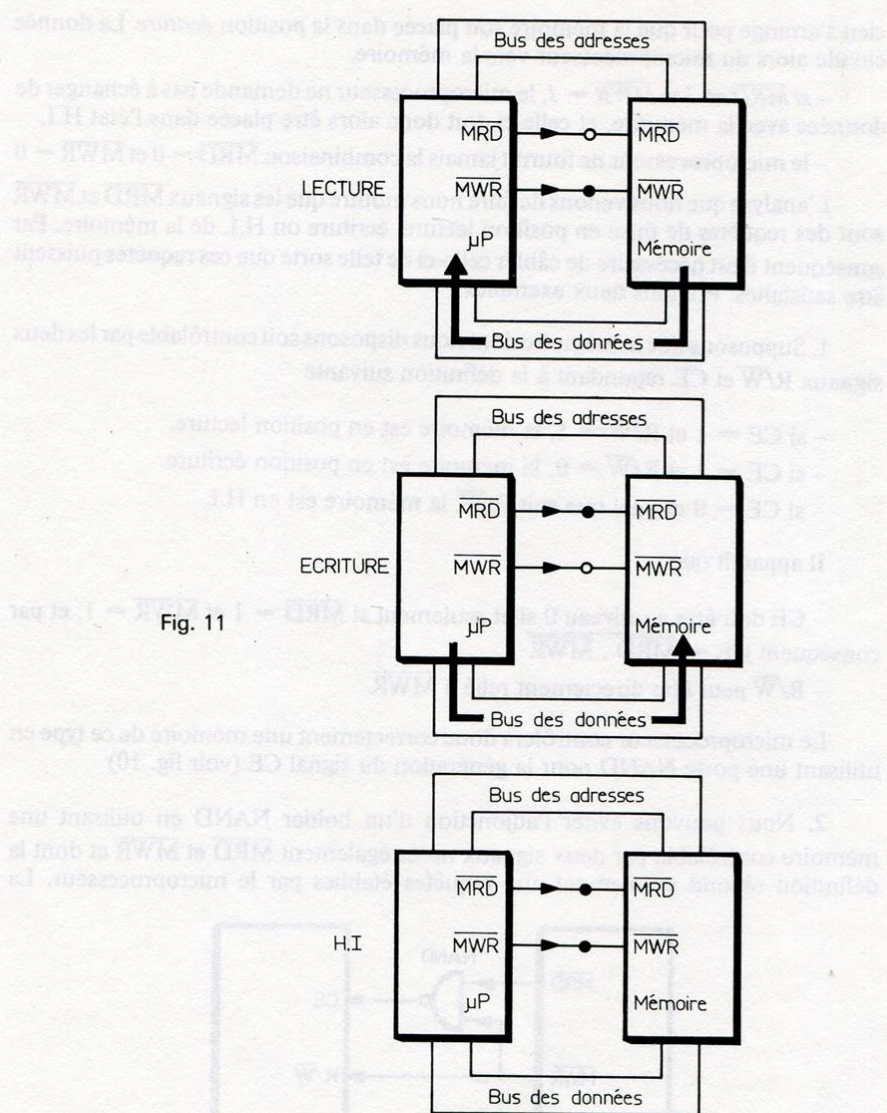


Fig. 11

Note:

- LED éteinte (niveau 0)
- LED allumée (niveau 1)

mémoire CDP 1823 utilisée sur notre maquette est de ce type et son contrôle s'effectue par liaison directe avec les signaux de lecture et d'écriture fournis par le microprocesseur.

Nous avons représenté figure 11 les trois états possibles de la mémoire ainsi que le sens de circulation des données.

Sur la maquette A, les signaux $\overline{\text{MRD}}$ et $\overline{\text{MWR}}$ sont visualisés par des LED ;

II - 3. Cycle de lecture et cycle d'écriture

Une lecture ou une écriture de donnée ne s'effectue pas instantanément. En effet, le microprocesseur doit avant tout adresser complètement la mémoire, ce qui nécessite d'attendre que le poids faible \mathcal{A}_0 soit présent sur le bus des adresses.

En outre, lors d'une opération de lecture, la donnée n'est pas disponible en sortie de la mémoire immédiatement après l'apparition de \mathcal{A}_0 , mais seulement au bout d'un temps supérieur ou égal à $t_S + t_A$:

- t_S (Set up time) est le temps de réponse du décodeur d'adresses interne au boîtier mémoire. Ceci veut dire que la mémoire proprement dite (décodeur exclu) est effectivement adressée au bout d'un temps t_S après l'apparition de \mathcal{A}_0 sur le bus d'adresses.

- t_A est le temps d'accès de la mémoire proprement dite, c'est-à-dire le temps de réponse de chaque registre latch constituant une cellule.

Remarque : par la suite, lorsque nous parlerons de temps d'accès, nous conviendrons qu'il s'agit du temps d'accès de la mémoire, décodeur inclus, c'est-à-dire de la somme $t_S + t_A$.

De la même façon, une donnée destinée à être écrite dans la mémoire n'est prise en compte qu'un certain temps après l'apparition de \mathcal{A}_0 .

Par ailleurs, les signaux de contrôle $\overline{\text{MRD}}$ et $\overline{\text{MWR}}$ doivent être générés par le microprocesseur selon une certaine chronologie dont le but est de respecter les caractéristiques dynamiques du constructeur de mémoires.

Pour ces raisons le microprocesseur effectue une opération de lecture ou d'écriture à l'intérieur d'un intervalle de temps égal à 8 périodes d'horloge et que nous avons déjà appelé cycle. Nous donnons figures 12 et 13 les chronogrammes relatifs respectivement à un cycle de lecture et à un cycle d'écriture. Ces diagrammes appellent quelques commentaires :

I. Cycle de lecture

Au début du cycle, $\overline{\text{MRD}}$ et $\overline{\text{MWR}}$ étant au niveau 1, la mémoire est en H.I. Puis le microprocesseur impose un niveau 0 sur $\overline{\text{MRD}}$ pour indiquer à la mémoire qu'il demande à la lire. A partir de l'instant d'apparition de \mathcal{A}_0 , la mémoire dispose au

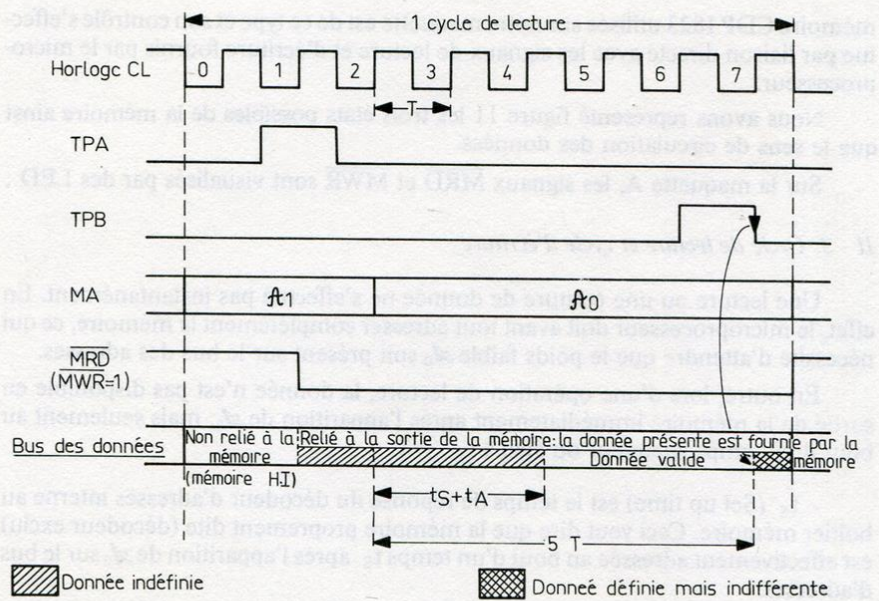


Fig. 12

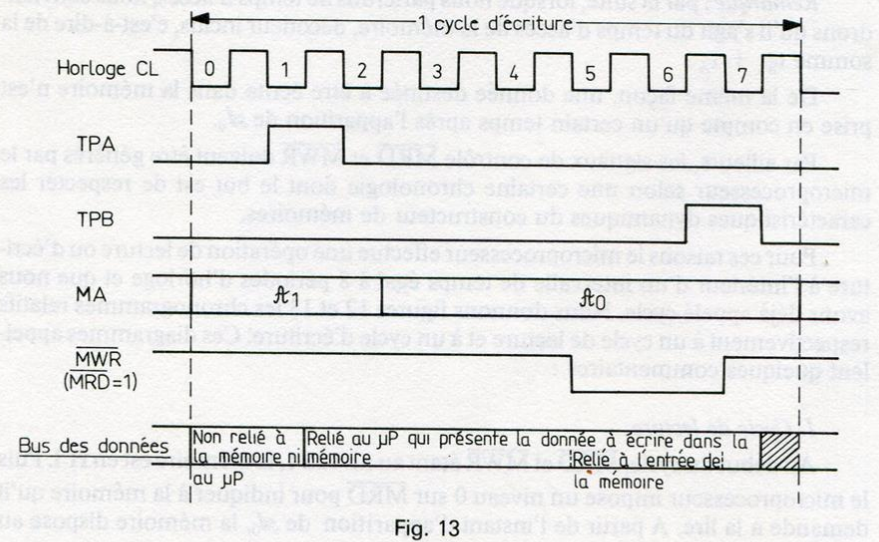


Fig. 13

maximum d'un intervalle de temps égal à 5 périodes d'horloge pour présenter une donnée valide sur le bus des données ; en effet, la donnée est mémorisée dans l'un des registres internes au microprocesseur seulement *lors du front descendant de TPB*.

2. Cycle d'écriture

Au début du cycle, $\overline{\text{MRD}}$ et $\overline{\text{MWR}}$ étant au niveau 1, la mémoire est en H.I. comme indiqué sur le diagramme, le microprocesseur est également en H.I. (il est déconnecté du bus des données). Puis le microprocesseur présente sur le bus des données une donnée qu'il demande à écrire dans la RAM en imposant un niveau 0 sur $\overline{\text{MWR}}$ après l'apparition de \mathcal{A}_0 .

III. L'accès direct à la mémoire

Il est tout à fait clair que le microprocesseur peut remplir de façon cohérente un projet déterminé seulement si la mémoire a été programmée à cet effet. L'utilisateur doit donc être autorisé à écrire une liste d'instructions (programme) dans la RAM et disposer par conséquent d'une voie directe d'accès à celle-ci de telle sorte qu'il puisse communiquer avec elle.

Il faut bien se rendre compte que l'écriture préalable d'un programme dans la mémoire, et son exécution par le microprocesseur sont deux fonctions tout à fait indépendantes. En général, lors de l'utilisation d'un système bâti autour d'un microprocesseur d'un autre type que le COSMAC, l'enregistrement d'une liste d'instructions dans la RAM nécessite des circuits complémentaires destinés à ménager une voie d'accès directe à la RAM. Dans ce cas, les deux procédés ordinairement utilisés sont les suivants :

1. Dans ce premier procédé, le microprocesseur est « déconnecté » du système.

Des entrées de contrôle sont prévues à cet effet et leur rôle est de :

- isoler le microprocesseur du bus des adresses, de telle sorte qu'un compteur auxiliaire puisse adresser séquentiellement la mémoire à partir de l'adresse initiale correspondant à la première instruction du programme à enregistrer.

- isoler le microprocesseur du bus des données, de telle sorte que le programmeur puisse présenter sur celui-ci les mots binaires destinés à être enregistrés dans la mémoire.

- isoler le microprocesseur des lignes de contrôle de la mémoire ; c'est un circuit annexe qui place celle-ci en position écriture.

La coordination des diverses opérations ayant pour but la programmation de la mémoire est assurée par une logique cablée à l'intérieur d'un tel système.

Le défaut évident d'une telle procédure est la lourdeur matérielle de l'ensemble ainsi réalisé.

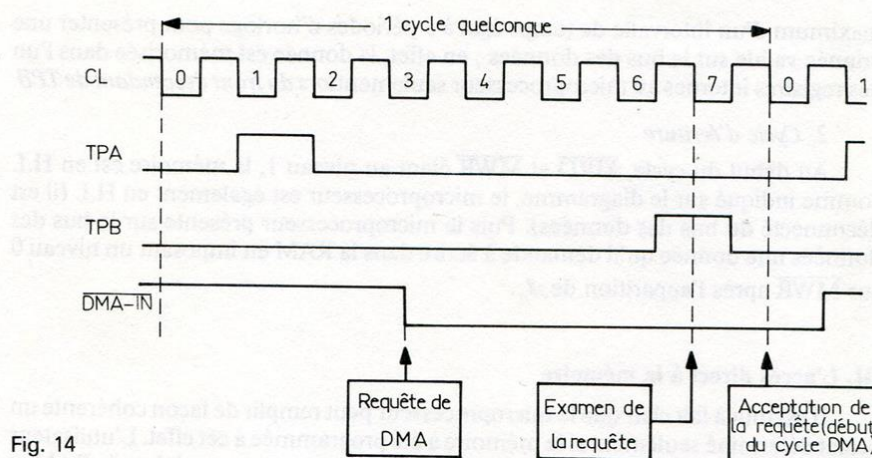


Fig. 14

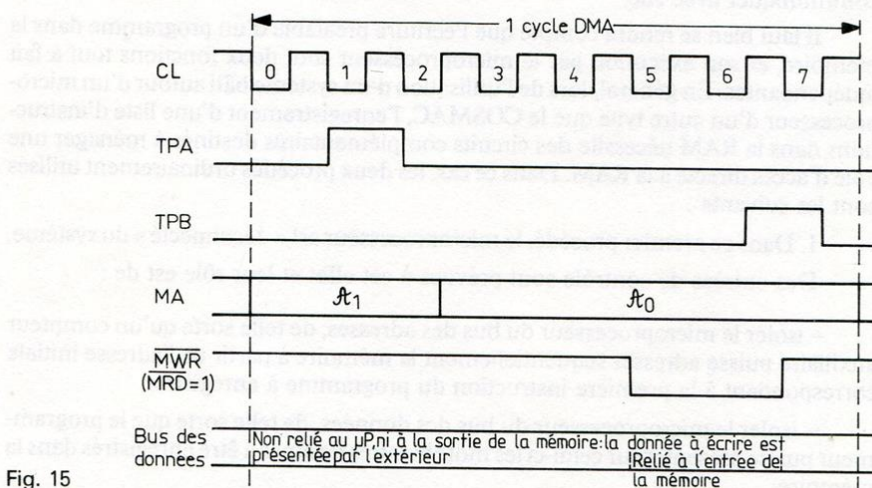


Fig. 15

2. L'idée de base du deuxième procédé est de demander au microprocesseur lui-même sa contribution dans l'enregistrement du programme dans la RAM. Un tel système inclut une mémoire morte ROM contenant toutes les instructions utiles à l'accès direct à la RAM ; dans ce cas, l'exécution par le microprocesseur du programme inscrit dans la ROM (appelé programme moniteur) a pour fonction d'auto-

riser l'utilisateur à enregistrer son programme dans la RAM. En outre, l'écriture d'une donnée est en général facilitée par la présence d'un clavier hexadécimal ; la prise en compte de l'enfoncement d'une touche ainsi que la scrutation du clavier sont effectués par le microprocesseur lors du déroulement du programme moniteur.

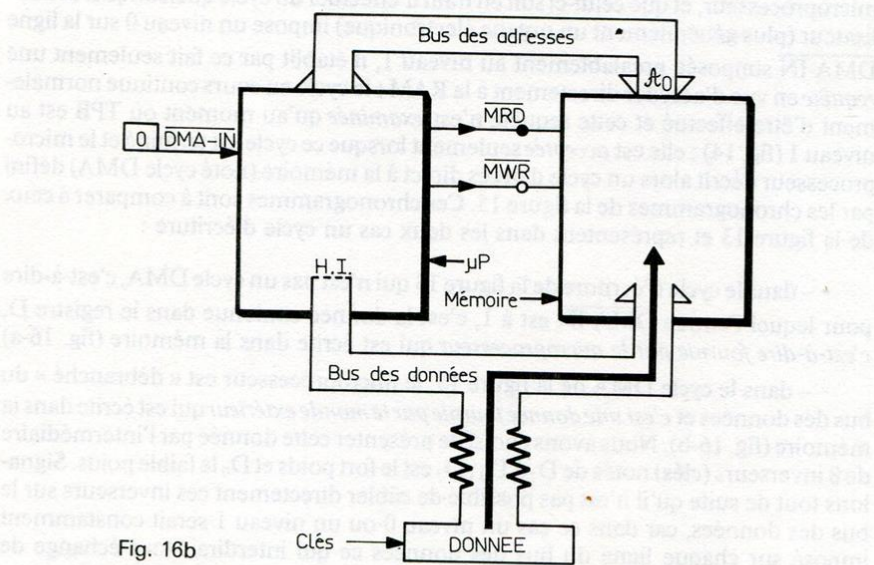
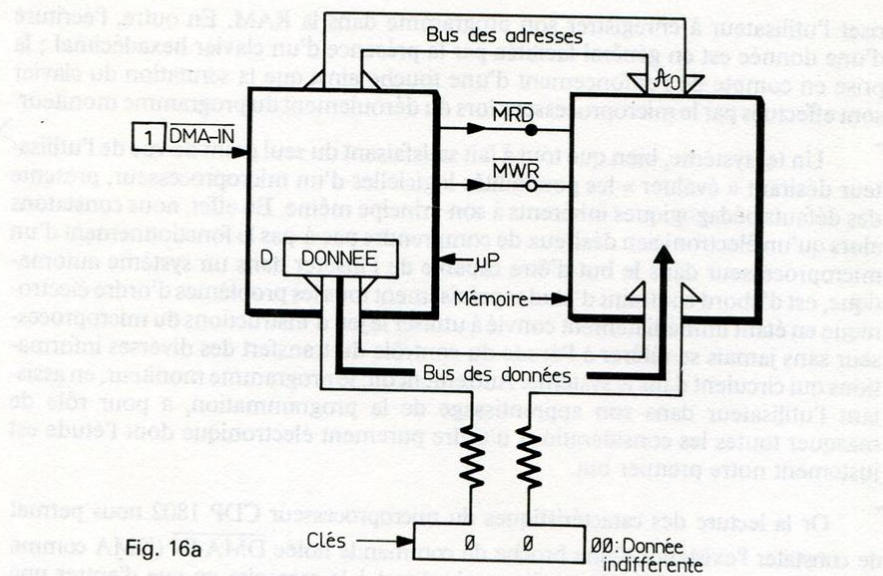
Un tel système, bien que tout à fait satisfaisant du seul point de vue de l'utilisateur désirant « évaluer » les possibilités logicielles d'un microprocesseur, présente des défauts pédagogiques inhérents à son principe même. En effet, nous constatons alors qu'un électronicien désireux de comprendre pas-à-pas le fonctionnement d'un microprocesseur dans le but d'être capable de l'insérer dans un système automatique, est d'abord contraint d'éluder précisément tous les problèmes d'ordre électronique en étant immédiatement convié à utiliser le jeu d'instructions du microprocesseur sans jamais se référer à l'étude du contrôle du transfert des diverses informations qui circulent dans le système. Autrement dit, le programme moniteur, en assistant l'utilisateur dans son apprentissage de la programmation, a pour rôle de masquer toutes les considérations d'ordre purement électronique dont l'étude est justement notre premier but.

Or la lecture des caractéristiques du microprocesseur CDP 1802 nous permet de constater l'existence d'une broche de commande notée $\overline{\text{DMA-IN}}$ (DMA comme Direct Memory Access, c'est-à-dire accès direct à la mémoire en vue d'entrer une donnée dans celle-ci). Quel est le rôle de cette entrée de commande ?

Supposons que des impulsions d'horloge soient fournies sur l'entrée CL du microprocesseur, et que celui-ci soit en train d'effectuer un cycle quelconque. Si l'utilisateur (plus généralement un organe électronique) impose un niveau 0 sur la ligne $\overline{\text{DMA-IN}}$ supposée préalablement au niveau 1, il établit par ce fait seulement une *requête* en vue d'accéder directement à la RAM ; le cycle en cours continue normalement d'être effectué et cette requête n'est *examinée* qu'au moment où TPB est au niveau 1 (fig. 14) ; elle est *acceptée* seulement lorsque ce cycle est terminé et le microprocesseur décrit alors un cycle d'accès direct à la mémoire (noté cycle DMA) défini par les chronogrammes de la figure 15. Ces chronogrammes sont à comparer à ceux de la figure 13 et représentent dans les deux cas un cycle d'écriture :

- dans le cycle d'écriture de la figure 13 qui n'est pas un cycle DMA, c'est-à-dire pour lequel l'entrée $\overline{\text{DMA-IN}}$ est à 1, c'est la donnée contenue dans le registre D, *c'est-à-dire fournie par le microprocesseur* qui est écrite dans la mémoire (fig. 16-a)

- dans le cycle DMA de la figure 15, le microprocesseur est « débranché » du bus des données et *c'est une donnée fournie par le monde extérieur* qui est écrite dans la mémoire (fig. 16-b). Nous avons choisi de présenter cette donnée par l'intermédiaire de 8 inverseurs (clés) notés de D_7 à D_0 ; D_7 est le fort poids et D_0 le faible poids. Signalons tout de suite qu'il n'est pas possible de câbler directement ces inverseurs sur le bus des données, car dans ce cas un niveau 0 ou un niveau 1 serait constamment imposé sur chaque ligne du bus des données ce qui interdirait tout échange de



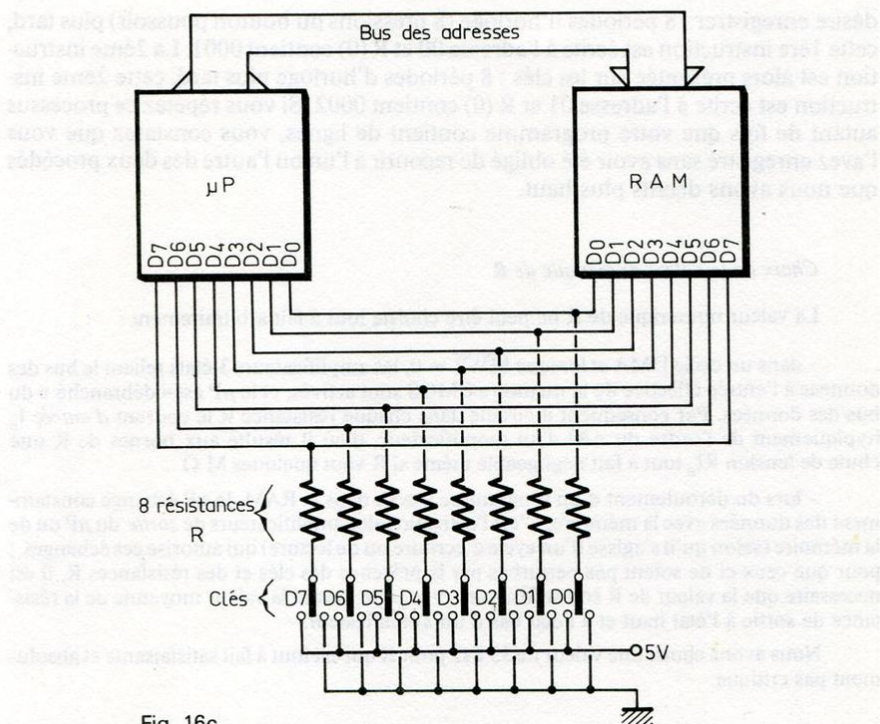


Fig. 16c

données entre le μP et la mémoire. La solution la plus simple consiste à insérer une résistance R entre l'inverseur D_i et la ligne D_i (fig. 16-c) ; nous discuterons plus loin de la valeur numérique à donner à R .

La prise en compte par le μP de la requête établie en imposant un niveau 0 sur l'entrée $\overline{DMA-IN}$ ne conduit pas seulement à placer la mémoire dans l'état d'emmagasiner une donnée présente sur les clés. En effet, le μP est microprogrammé de telle sorte qu'*au cours du déroulement d'un cycle DMA le registre $R(0)$ est pointeur de la mémoire* ; par ailleurs $R(0)$ s'incrmente automatiquement en fin du cycle DMA. L'ensemble des opérations effectuées au cours d'un tel cycle peut donc se représenter symboliquement ainsi :

$$\text{DONNÉE} \rightarrow M(R(0)) ; R(0) + 1$$

Après initialisation du μP (voir plus loin), $R(0)$ contient 0000 en notation hexadécimale et pointe donc la ligne d'adresse 00 de la mémoire ; si l'horloge du système est placée dans le mode « pas-à-pas » l'utilisateur dispose de tout le temps nécessaire pour présenter sur les clés D_7 à D_0 la 1ère instruction du programme qu'il

désire enregistrer ; 8 périodes d'horloge (8 pressions du bouton poussoir) plus tard, cette 1ère instruction est écrite à l'adresse 00 et R (0) contient 0001. La 2ème instruction est alors présentée sur les clés ; 8 périodes d'horloge plus tard, cette 2ème instruction est écrite à l'adresse 01 et R (0) contient 0002. Si vous répétez ce processus autant de fois que votre programme contient de lignes, vous constatez que vous l'avez enregistré sans avoir été obligé de recourir à l'un ou l'autre des deux procédés que nous avons décrits plus haut.

Choix de la valeur numérique de R

La valeur numérique de R ne peut être choisie tout à fait arbitrairement :

- dans un cycle DMA et lorsque $\overline{MWR} = 0$, les amplificateurs 3-états reliant le bus des données à l'entrée effective de la mémoire CMOS sont activés, et le μP est « débranché » du bus des données. Par conséquent il circule dans chaque résistance R le courant d'entrée I_e (typiquement de l'ordre du nA) d'un amplificateur. d'où il résulte aux bornes de R une chute de tension RI_e tout à fait négligeable même si R vaut quelques M Ω

- lors du déroulement d'un programme inscrit dans la RAM, le μP échange constamment des données avec la mémoire. C'est l'activation des amplificateurs de sortie du μP ou de la mémoire (selon qu'il s'agisse d'un cycle d'écriture ou de lecture) qui autorise ces échanges ; pour que ceux-ci ne soient pas perturbés par la présence des clés et des résistances R, il est nécessaire que la valeur de R soit suffisamment grande devant la valeur moyenne de la résistance de sortie à l'état haut et à l'état bas d'un amplificateur.

Nous avons choisi une valeur de 33 k Ω pour R qui est tout à fait satisfaisante et absolument pas critique.

IV. Les « interruptions » de programme ou, « comment dialoguer avec le système »

Nous venons de voir que l'utilisateur peut enregistrer un programme dans la RAM en agissant sur l'entrée $\overline{DMA-IN}$ du μP . Rappelons qu'au cours du chargement de la mémoire, le μP n'est pas dans son mode de fonctionnement normal qui est d'exécuter ce programme ; cependant rien n'interdit d'établir une requête en vue d'obtenir un accès direct à la mémoire *au cours de l'exécution d'un programme*. Dans ce cas, dès l'acceptation de la requête, c'est-à-dire dès que le cycle dans lequel se trouve le μP est terminé, le déroulement du programme s'interrompt : le μP décrit alors un cycle DMA pendant lequel une donnée présentée sur les clés D_7 à D_0 est écrite dans la mémoire à l'adresse contenue dans le registre R (0). Dès la fin de ce cycle et si une nouvelle requête n'est pas établie, le μP reprend le cours normal de ses opérations en continuant d'exécuter le programme de l'endroit où il l'a laissé. Le système μP -mémoire s'est enrichi d'une donnée nouvelle.

Il y a une autre façon d'interrompre un programme en cours d'exécution ; en effet, le μP présente une entrée notée $\overline{INTERRUPT}$ dont la mise au niveau 0 établit

ce qu'on appelle une requête d'interruption. Contrairement à la requête DMA, son acceptation est conditionnée au contenu d'un registre interne au μP de capacité 1 bit et noté IE (comme « Interrupt Enable » c'est-à-dire « autorisation d'interruption »). Si $IE = 1$, l'autorisation est en règle et le μP interrompt alors l'exécution du programme en cours en entamant une procédure dont nous étudierons plus loin le détail. Disons ici seulement que cette procédure aboutit à permettre l'instauration d'un dialogue entre l'utilisateur et le système μP -mémoire ; en effet, comme dans le mode DMA, une donnée *peut* alors être introduite dans le système par l'intermédiaire des clés mais pas nécessairement à une adresse définie par le contenu de R (0).

Remarques :

1. L'entrée $\overline{DMA-OUT}$ du μP que nous n'avons pas jugé utile d'utiliser sur la maquette A (elle est donc constamment au niveau 1) a une fonction tout à fait semblable à l'entrée $\overline{DMA-IN}$. En effet, si vous imposez un niveau 0 sur $\overline{DMA-OUT}$ vous établissez également une requête d'accès direct à la mémoire mais cette fois en vue de la lire ; les opérations réalisées au cours d'un cycle $\overline{DMA-OUT}$ sont donc les suivantes :

$$M(R(0)) \longrightarrow \text{Bus des données ; } R(0) + 1$$

Cette procédure permet de décharger une RAM vers le monde extérieur dans le but par exemple de sauvegarder son contenu dans une mémoire morte ou de l'afficher sur un écran cathodique. C'est parce que ces applications n'entrent pas dans le cadre de cet ouvrage que nous n'avons pas prévu la possibilité de contrôler l'entrée $\overline{DMA-OUT}$.

2. Si au cours d'un cycle quelconque, on établit une requête d'interruption (entrée INTERRUPT) *et* une requête de DMA (entrée $\overline{DMA-IN}$), seul l'accès direct à la mémoire est accepté (la requête DMA est prioritaire).

Sur la maquette A, les entrées $\overline{DMA-IN}$ et INTERRUPT sont commandées par deux inverseurs destinés à être utilisés lorsque l'horloge du système est dans le mode « pas-à-pas ».

V. Les signaux d'état

Nous venons de voir que l'utilisateur peut interrompre le cours normal du déroulement d'un programme en établissant une requête en vue d'obtenir un service ; ce service consiste soit en l'établissement d'une voie d'accès direct à la mémoire soit en l'exécution d'une procédure spéciale d'interruption. Le μP indique qu'il accepte de rendre ce service en envoyant sur les deux lignes SC1 et SC0 (SC comme « State Control », c'est-à-dire signaux de contrôle de l'état dans lequel se trouve le μP) tels que :

– Si $SC1 = 1$ et $SC0 = 0$, le μP signale par ce fait qu'il est dans *un cycle DMA* ; la voie directe d'accès est effectivement aménagée.

– Si $SC1 = 1$ et $SC0 = 1$, le μP signale par ce fait qu'il est dans un cycle d'interruption et qu'il accepte donc la requête qui a été établie en imposant un niveau 0 sur son entrée $\overline{INTERRUPT}$.

Si aucune de ces deux requêtes n'est établie, le μP est en mode de fonctionnement normal, c'est-à-dire qu'il est en train d'exécuter un programme préalablement enregistré dans la mémoire. Or nous avons déjà vu dans la première partie de cet ouvrage que l'exécution de chaque instruction du programme nécessitait évidemment la recherche préalable de son code dans la mémoire. Il nous apparaît ainsi que lors du déroulement normal d'un programme, le μP est alternativement dans l'un des deux états suivants :

– Si $SC1 = 0$ et $SC0 = 0$, le μP signale par ce fait qu'il est en train de rechercher dans la mémoire le code de l'instruction qu'il doit exécuter : il est dans un cycle de recherche

– Si $SC1 = 0$ et $SC0 = 1$, le μP signale par ce fait qu'il est en train d'exécuter effectivement l'instruction dont il vient de lire le code au cours du cycle de recherche : il est dans un cycle d'exécution.

Sur la maquette A, les signaux $SC1$ et $SC0$ sont visualisés par des LED de telle sorte que l'utilisateur, utilisant le système dans le mode « pas-à-pas », puisse constamment contrôler dans quel état se trouve le μP .

On peut se demander quel est le rôle des signaux d'état $SC1$ et $SC0$. En effet, le constructeur n'a sûrement pas décidé de leur existence dans le seul but de nous permettre d'y placer des LED et par suite de nous assurer un contrôle permanent du fonctionnement correct du μP dans le mode pas-à-pas. Les soucis du constructeur ne sont pas d'ordre pédagogique ; en réalité les signaux $SC1$ et $SC0$ sont destinés à commander des organes électroniques qui ont besoin de savoir dans quel état se trouve le μP . Dans quel but ?

Sur notre maquette, toute requête en vue d'obtenir une interruption de programme est établie en imposant un niveau 0 sur l'entrée $\overline{DMA-IN}$ ou l'entrée $\overline{INTERRUPT}$; ceci est réalisé par l'intermédiaire des inverseurs câblés sur ces entrées, et l'étude expérimentale des procédures qui en découlent nécessite que l'horloge soit placée dans le mode « astable » et réglée, pour fixer les idées, à 1 MHz ; supposez en outre qu'un programme que vous avez préalablement enregistré dans la RAM soit en train de se dérouler, et que votre projet soit d'établir une requête en vue d'écrire la donnée \mathcal{D} dans la mémoire à l'adresse \mathcal{A}_0 définie par le contenu de $R(0)$. Vous présentez alors \mathcal{D} sur les clés. Si vous imposez un niveau 0 sur l'entrée $\overline{DMA-IN}$ en utilisant l'inverseur associé, 8 μs après acceptation de la requête vous obtenez effectivement le résultat escompté c'est-à-dire que \mathcal{D} est bien enregistré à l'adresse \mathcal{A}_0 ; mais comme pendant ce cycle DMA vous n'avez pas eu le temps matériel de remettre

l'entrée $\overline{\text{DMA-IN}}$ au niveau 1, le μP décrit un cycle DMA supplémentaire puis un autre et ceci tant que $\overline{\text{DMA-IN}}$ est au niveau 0. Au bout du compte, puisque R (0) s'incrémente à chaque cycle DMA, vous avez enregistré \mathcal{D} non seulement à l'adresse \mathcal{A}_0 , mais aux adresses $\mathcal{A}_0 + 01$, $\mathcal{A}_0 + 02$, etc... IL suffit que $\overline{\text{DMA-IN}}$ reste au niveau 0 pendant $256 \times 8 \mu\text{s}$ soit environ 2ms, pour que votre mémoire soit entièrement remplie par \mathcal{D} ; votre programme s'est donc effacé et plus rien ne marche. Pour éviter cet inconvénient, il faut confier à un organe électronique le soin d'établir une requête d'interruption. A cet effet, la maquette A comporte une bascule JK CD 4027 dont la sortie $\overline{\text{Q}}$ notée S.R. (« Service Request » ou Requête en vue d'obtenir un service) est susceptible de commander l'entrée $\overline{\text{DMA-IN}}$ ou $\overline{\text{INTERRUPT}}$ du μP . (fig. 17). La sortie SC1 du μP est reliée à l'entrée de forçage RESET de la bascule, et son déclenchement est assuré par un front montant généré par une pression du bouton poussoir.

Supposez que $\overline{\text{S.R.}}$ soit relié à $\overline{\text{DMA-IN}}$ et que comme précédemment vous désiriez, au cours du déroulement d'un programme, écrire la donnée \mathcal{D} à l'adresse

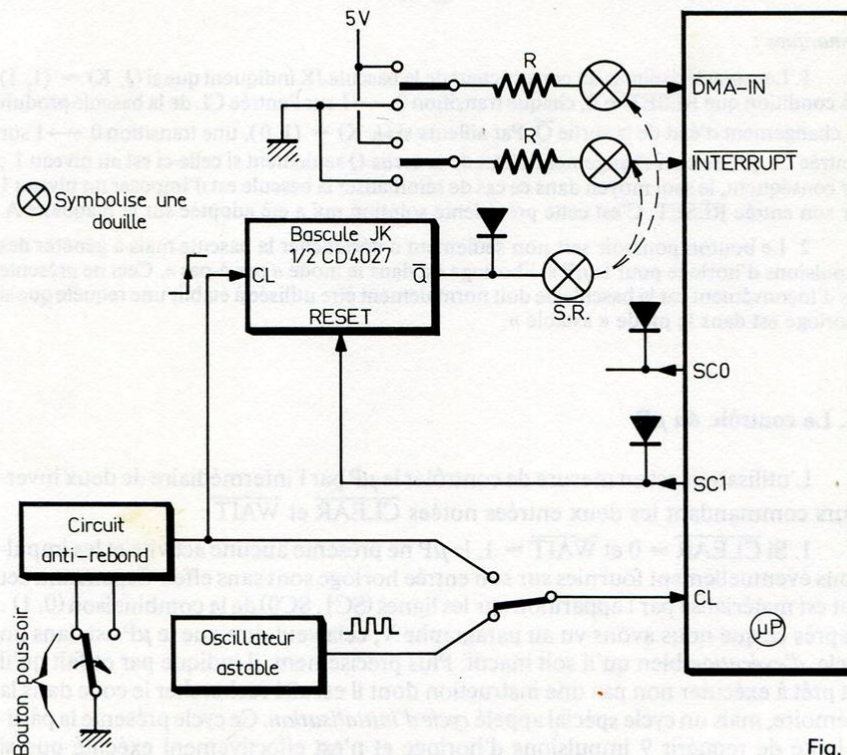


Fig. 17

\mathcal{A}_0 définie par le contenu de $R(0)$. Si $\overline{S.R.}$ est initialement au niveau 1, une pression du bouton poussoir va déclencher la bascule ce qui amènera $\overline{S.R.} = \overline{DMA-IN}$ au niveau 0 : la requête est établie (fig. 18). Le μP indique qu'il s'engage dans le cycle DMA en envoyant un niveau 1 sur la ligne SC1 ce qui a pour effet de réinitialiser la bascule ($\overline{S.R.}$ revient à 1). Par conséquent, un cycle DMA et un seul se déroule ce qui est le résultat recherché.

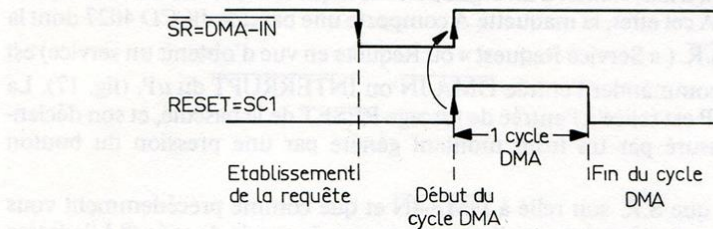


Fig. 18

Remarques :

1. Les caractéristiques du constructeur de la bascule JK indiquent que si $(J, K) = (1, 1)$ et à condition que $RESET = 0$, chaque transition $0 \rightarrow 1$ sur l'entrée CL de la bascule produit un changement d'état de la sortie \overline{Q} . Par ailleurs si $(J, K) = (1, 0)$, une transition $0 \rightarrow 1$ sur l'entrée CL produit un changement d'état de la sortie Q seulement si celle-ci est au niveau 1 ; par conséquent, le seul moyen dans ce cas de réinitialiser la bascule est d'imposer un niveau 1 sur son entrée RESET. C'est cette précédente solution qui a été adoptée sur la maquette A.

2. Le bouton poussoir sert non seulement à déclencher la bascule mais à générer des impulsions d'horloge pour le μP si l'horloge est dans le mode « pas-à-pas ». Ceci ne présente pas d'inconvénient car la bascule ne doit normalement être utilisée à établir une requête que si l'horloge est dans le mode « astable ».

VI. Le contrôle du μP

L'utilisateur est en mesure de contrôler le μP par l'intermédiaire de deux inverseurs commandant les deux entrées notées \overline{CLEAR} et \overline{WAIT} :

1. Si $\overline{CLEAR} = 0$ et $\overline{WAIT} = 1$, le μP ne présente aucune activité et les impulsions éventuellement fournies sur son entrée horloge sont sans effet. Cependant, cet état est matérialisé par l'apparition sur les lignes (SC1, SC0) de la combinaison (0, 1) : d'après ce que nous avons vu au paragraphe V, cela veut dire que le μP est dans un cycle d'exécution bien qu'il soit inactif. Plus précisément, il indique par ce fait qu'il est prêt à exécuter non pas une instruction dont il est allé rechercher le code dans la mémoire, mais un cycle spécial appelé cycle d'initialisation. Ce cycle présente la particularité de requérir 9 impulsions d'horloge et n'est effectivement exécuté que si

l'entrée $\overline{\text{CLEAR}}$ est remise au niveau 1. Nous verrons plus loin le détail de la procédure que développe alors le μP ; indiquons cependant qu'elle aboutit essentiellement au résultat suivant : *les contenus des registres P et R (0) sont mis à zéro*. Or, rappelons que le registre P est pointeur du compteur de programme ; par conséquent *le compteur de programme est nécessairement $R(P) = R(0)$ après un cycle d'initialisation et son contenu est égal à zéro*.

2. Si $\overline{\text{CLEAR}} = 1$ et $\overline{\text{WAIT}} = 1$ et après le déroulement du cycle d'initialisation, le μP « tourne normalement », c'est-à-dire qu'il exécute effectivement, au rythme de l'horloge, le programme inscrit dans la mémoire à moins qu'il ne développe une procédure d'interruption ou qu'il n'aménage une voie d'accès direct à la RAM en réponse à une requête éventuellement établie. Pour éviter des complications de langage, nous dirons que si $\overline{\text{CLEAR}} = 1$ et $\overline{\text{WAIT}} = 1$, le μP est dans le mode RUN (du verbe anglais « to run »)

3. Si pendant le mode RUN du μP on impose un niveau 0 sur l'entrée $\overline{\text{WAIT}}$, ce qui aboutit à la combinaison $\overline{\text{CLEAR}} = 1$ et $\overline{\text{WAIT}} = 0$, le μP stoppe son activité en ignorant les impulsions d'horloge arrivant sur son entrée CL après la transition $1 \rightarrow 0$ de $\overline{\text{WAIT}}$. Le contenu de tous ses registres internes n'est pas affecté. Le μP est alors dans un état d'attente et effectue une pause dont l'utilité n'est pas exploitée sur la maquette A mais qui peut cependant être comprise en considérant l'exemple suivant : supposons que le μP soit en train d'effectuer un cycle de lecture à la fréquence 2MHz ; le temps d'accès de la mémoire doit donc être inférieur à 250 ns (voir paragraphe II - 3). Si cela n'est pas le cas, une lecture correcte ne peut s'opérer qu'en imposant un niveau 0 sur l'entrée $\overline{\text{WAIT}}$ en vue de mettre le μP dans l'état d'attendre que la mémoire soit prête à fournir sa donnée.

4. Si $\overline{\text{CLEAR}} = 0$ et $\overline{\text{WAIT}} = 0$, le μP développe une procédure spéciale de chargement de la RAM dont nous parlerons au paragraphe VIII.

VII. Les communications avec le monde extérieur

VII. 1. Les communications directes

Appelons \mathcal{S} le système constitué par le μP et la mémoire. Ce système ne peut travailler de façon cohérente que si la mémoire a été chargée par un programme. A cet effet nous avons vu qu'il suffisait de demander au μP d'exécuter une succession de cycles DMA en vue d'enregistrer la suite des codes des instructions présentés sur les clés D_7 à D_0 . Par ailleurs, une donnée présentée sur ces clés peut être introduite dans le système \mathcal{S} au cours du déroulement d'un programme, et ceci grâce aux procédures d'interruption. Par conséquent vous voyez que le système \mathcal{S} n'est pas totalement isolé du monde extérieur puisque vous pouvez lui fournir des informations par l'intermédiaire des clés.

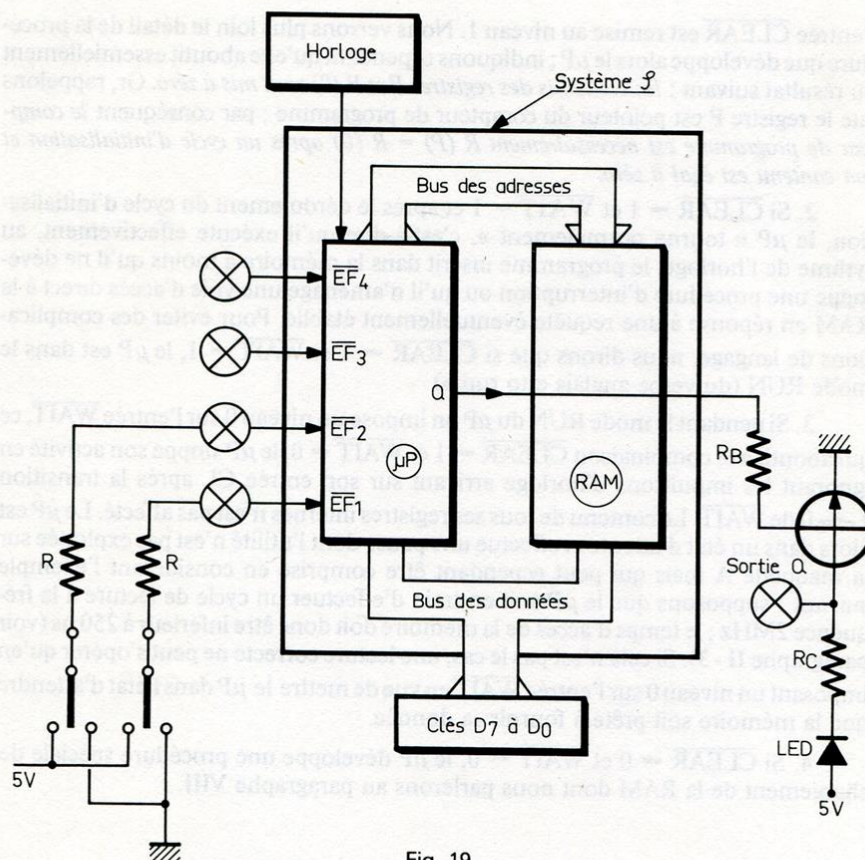


Fig. 19

Cependant cette communication est à sens unique, et le système \mathcal{S} n'offrirait aucun intérêt pratique s'il n'était en mesure de fournir au monde extérieur le résultat utile de son travail. Le μP présente à cet effet une sortie Q programmable, directement accessible à tout organe électronique dont l'utilisateur serait désireux d'assurer la commande. Sur la maquette A, cette sortie Q est visualisée par une LED ; le collecteur du transistor alimentant la diode est relié à une douille disponible à tout dispositif dont la consommation n'excède pas 100 mA (la sortie Q du μP peut débiter 1 mA et on table sur un β minimum de 100 pour le transistor).

Par ailleurs le μP présente 4 entrées notées \overline{EF}_1 , \overline{EF}_2 , \overline{EF}_3 , \overline{EF}_4 , directement accessibles à tout capteur fournissant des informations concernant par exemple l'état d'un ensemble électronique commandé par la sortie Q. L'état logique de ces 4

entrées \overline{EF} est testé par programme. Sur la maquette A, ces entrées sont accessibles par des douilles, et deux d'entre elles (\overline{EF}_1 et \overline{EF}_2) peuvent également être commandées par des inverseurs.

Le système \mathcal{S} constitue ainsi un automate programmable à μP à 12 entrées (\overline{EF}_1 à \overline{EF}_4 et D_7 à D_0) et à une sortie Q, prêt à fonctionner (voir fig. 19).

Bien que le rôle de toutes les broches du μP ne soit pas encore défini, nous invitons le lecteur à s'attaquer dès à présent à l'étude des principes fondamentaux de la programmation des μP .

Nous pensons que la maquette A facilite cette étude et qu'elle la rend très attrayante ; par conséquent nous prions le lecteur désireux d'utiliser la maquette A de s'engager dans sa construction.

VII. 2. Le couplage du système \mathcal{S} avec le monde extérieur

Lors d'un cycle de lecture, le μP intime à la RAM l'ordre (matérialisé par la mise au niveau 0 de \overline{MRD}) de présenter un mot binaire de 8 bits sur le bus des données ; cet octet est ensuite mémorisé, en vue de son traitement, dans un registre *interne au μP* lors du front descendant de TPB.

Or il est facile d'imaginer que cet octet puisse être mémorisé, en vue de son accessibilité au monde extérieur, dans un registre *externe au μP* lors du front descendant de TPB. En effet, il suffit pour cela d'utiliser le signal TPB fourni par le μP pour déclencher le registre externe. Pour la maquette B nous avons choisi le registre latch CDP 1852 (RCA) dont l'entrée de déclenchement CL est telle que :

- si $CL = 1$, $DO = DI$ (D Output = D Input ; voir fig. 20)
- si CL passe au niveau 0, la donnée DO présente lorsque CL était au niveau 1 est mémorisée.

Les 8 lignes constituant le bus de sortie DO du registre externe sont 8 sorties destinées à commander divers organes électroniques par l'intermédiaire, sur la maquette B, de douilles connectées aux collecteurs de 8 transistors assurant par ailleurs la commande de LED de visualisation du niveau logique de chaque sortie. Nous nous apercevons ainsi que le registre externe multiplie les possibilités de l'automate programmable de la figure 19 en portant le nombre de ses sorties à un total de 9. Le registre externe est appelé « coupleur » ou circuit d'« interface » ; ces termes indiquent qu'une liaison est assurée entre deux systèmes qui sont dans notre cas le système \mathcal{S} et le système commandé par les sorties DO.

Cependant, la liaison entre le système \mathcal{S} et le monde extérieur doit être assurée seulement lorsque l'utilisateur le désire et non pas à chaque cycle de lecture comme notre exposé nous laisse actuellement le présumer. A cet effet, le coupleur présente une entrée CS invalidant son déclenchement si placée au niveau 0 ; par conséquent

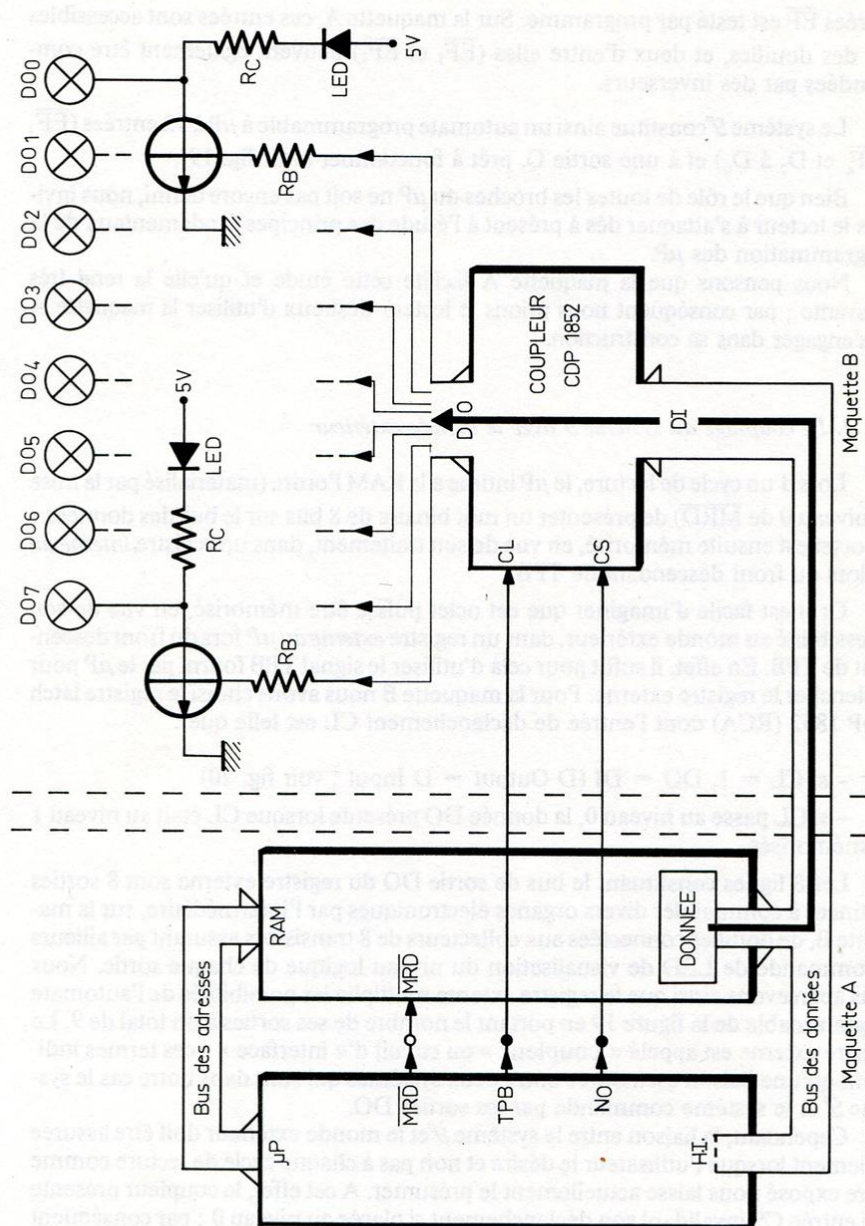


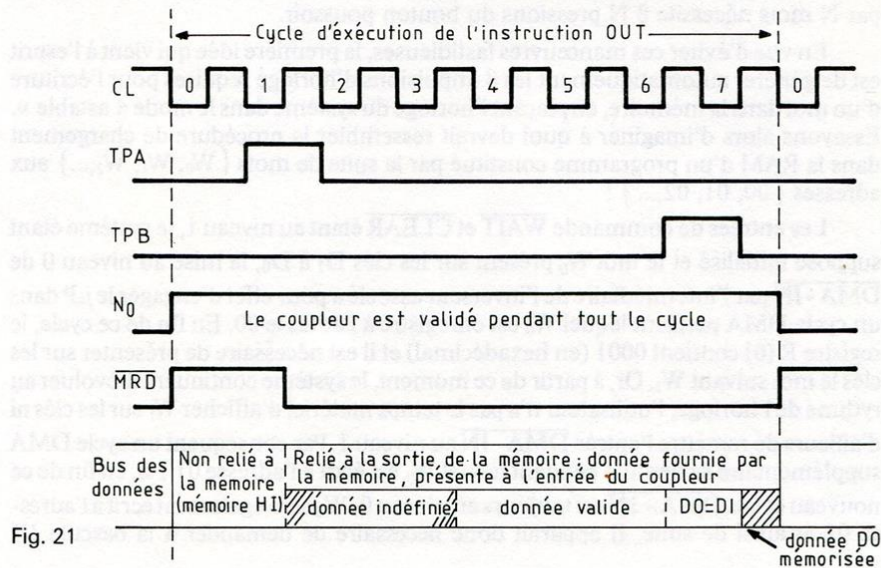
Fig. 20

la sortie d'une donnée ne pourra s'effectuer qu'en imposant un niveau 1 sur l'entrée CS au cours d'un cycle de lecture. C'est le μP qui est chargé de fournir ce niveau par l'intermédiaire de l'une de ses 3 sorties notées N_0, N_1, N_2 (sur la maquette B, c'est N_0 qui est relié à l'entrée CS du coupleur ; N_1 et N_2 ne sont pas utilisés). A ces 3 sorties sont associées 7 combinaisons binaires utiles (8 combinaisons moins la combinaison 000) : nous dirons que les 3 lignes N « sont à i » si i est le nombre décimal correspondant au nombre binaire ($N_2 N_1 N_0$) ; par exemple, si ($N_2 N_1 N_0$) = (101), les lignes N sont à 5. C'est lors de l'exécution de l'une des 7 instructions de sortie notées de OUT 1 à OUT 7 que le μP activera les 3 lignes N ; par exemple, pendant l'exécution de l'instruction OUT 1, N_0 passe au niveau 1 ce qui valide le coupleur : la donnée présentée par la RAM est mémorisée dans le registre externe lors du front descendant de TPB et est ainsi disponible pour une durée quelconque sur le bus de sortie DO.

Nous donnons figure 21 les chronogrammes relatifs à la sortie d'une donnée. Remarquez qu'il s'agit d'un cycle de *lecture* pendant lequel la mémoire fournit la donnée non pas au microprocesseur mais au coupleur. La figure 20 représente le transfert de la donnée lorsque le coupleur est transparent.

Remarque :

Parmi les divers types de μP , certains ne disposent pas de lignes N et par conséquent leur jeu d'instruction ne comporte pas d'instructions de sortie. L'interfaçage avec le monde extérieur est alors réalisé en utilisant des lignes du bus d'adresses pour valider le coupleur, celui-ci est par suite contrôlé par le μP exactement de la même façon que la mémoire. Dans ce cas, l'instruction de stockage est équivalente à une instruction de sortie si le stockage est effectué à l'adresse du coupleur. Remarquez que la sortie d'une donnée est alors une opération d'*écriture*.



VIII. Le mode de fonctionnement « LOAD » du μ P CDP 1802

VIII. 1. Une procédure spéciale de chargement d'un programme dans la mémoire

La possibilité qu'offre le système actuellement décrit de marcher à la fréquence 0 (en pas-à-pas) constitue un avantage pédagogique certain puisqu'alors son fonctionnement peut être dépouillé de tout le mystère généralement associé aux ensembles à μ P.

Cependant, dès lors que l'utilisateur a développé complètement l'étude détaillée des mécanismes intimes élaborés par le μ P et qui assurent le transfert et le traitement des informations circulant dans le système, il lui est loisible de décider de ne plus s'y intéresser (quitte à y revenir ultérieurement) et donc de fixer son attention seulement sur l'aspect global du fonctionnement de l'ensemble μ P – mémoire – coupleur.

En particulier, la mise au niveau 0 de $\overline{\text{DMA}} - \overline{\text{IN}}$ engage le μ P dans un cycle DMA pendant lequel le mot présent sur les clés D_7 à D_0 est écrit dans la RAM ; si ce mot est une instruction appartenant au programme que l'on veut enregistrer, les 8 impulsions d'horloge nécessaires au déroulement du cycle DMA sont normalement fournies par 8 pressions successives du bouton poussoir pendant lesquelles l'utilisateur est en mesure de détailler la procédure de chargement de la donnée dans la mémoire. Dès lors que cette procédure est complètement élucidée, la faculté qu'offre le système de fonctionner en pas-à-pas n'a plus de raison d'être exploitée; par ailleurs il apparaît que l'enregistrement dans la mémoire d'un programme constitué par N mots nécessite 8 N pressions du bouton poussoir.

En vue d'éviter ces manœuvres fastidieuses, la première idée qui vient à l'esprit est de générer automatiquement les 8 impulsions d'horloge requises pour l'écriture d'un mot dans la mémoire, en plaçant l'horloge du système dans le mode « astable ». Essayons alors d'imaginer à quoi devrait ressembler la procédure de chargement dans la RAM d'un programme constitué par la suite de mots $\{W_0, W_1, W_2, \dots\}$ aux adresses $\{00, 01, 02, \dots\}$:

Les entrées de commande $\overline{\text{WAIT}}$ et $\overline{\text{CLEAR}}$ étant au niveau 1, le système étant supposé initialisé et le mot W_0 présent sur les clés D_7 à D_0 , la mise au niveau 0 de $\overline{\text{DMA}} - \overline{\text{IN}}$ par l'intermédiaire de l'inverseur associé a pour effet d'engager le μ P dans un cycle DMA pendant lequel W_0 est enregistré à l'adresse 00. En fin de ce cycle, le registre R(0) contient 0001 (en hexadécimal) et il est nécessaire de présenter sur les clés le mot suivant W_1 . Or, à partir de ce moment, le système continuant d'évoluer au rythme de l'horloge, l'utilisateur n'a pas le temps matériel d'afficher W_1 sur les clés ni d'ailleurs de remettre l'entrée $\overline{\text{DMA}} - \overline{\text{IN}}$ au niveau 1. Par conséquent un cycle DMA supplémentaire se déroule pendant lequel W_0 est écrit à l'adresse 01 ; si, en fin de ce nouveau cycle, $\overline{\text{DMA}} - \overline{\text{IN}}$ est toujours au niveau 0, W_0 sera également écrit à l'adresse 02 et ainsi de suite. Il apparaît donc nécessaire de demander à la bascule JK

(fig. 17) de commander l'entrée $\overline{\text{DMA}} - \overline{\text{IN}}$ du μP ; en effet, dès l'acceptation de la requête établie par une pression du bouton poussoir, la bascule est réinitialisée par le passage au niveau 1 de SC 1 (fig. 18). Un cycle DMA et un seul a donc lieu, ce qui évite l'enregistrement de W_0 à une adresse différente de 00.

Cependant, bien que $\overline{\text{DMA}} - \overline{\text{IN}}$ soit automatiquement remis à 1, l'utilisateur n'a toujours pas le temps matériel d'afficher W_1 sur les clés car, $\overline{\text{CLEAR}}$ et $\overline{\text{WAIT}}$ étant au niveau 1 et aucune requête d'interruption n'étant adressée au μP , celui-ci continue d'évoluer en *exécutant* le programme (éventuellement incohérent) écrit à partir de l'adresse 01 de la RAM. La seule façon de disposer de tout le temps nécessaire à l'affichage de W_1 sur les clés est de placer le μP dans un *état d'attente* pendant lequel il convient que le contenu du registre pointeur R (0) soit stable et qu'aucune opération d'écriture dans la mémoire ne soit effectuée. Par ailleurs, le μP doit être en mesure de quitter cet état d'attente dès l'établissement de la requête en vue de l'engager dans le cycle DMA suivant. Dans ces conditions, W_1 étant présent sur les clés, la mise au niveau 0 de $\overline{\text{DMA}} - \overline{\text{IN}}$ par l'intermédiaire du bouton poussoir aura pour effet d'enregistrer W_1 à l'adresse 01 et d'autoriser l'utilisateur à afficher sur les clés le mot suivant W_2 dont l'écriture à l'adresse 02 sera assurée par une nouvelle pression du bouton poussoir et ainsi de suite.

Il se trouve que la mise en œuvre de cette procédure ne nécessite point le concours de composants supplémentaires. En effet, comme nous l'avons laissé entendre au paragraphe VI, la mise au niveau 0 des *deux* entrées $\overline{\text{CLEAR}}$ et $\overline{\text{WAIT}}$ a pour effet d'assurer au μP un mode de fonctionnement propre à remplir précisément les conditions que nous venons de définir. L'électronicien désireux d'exploiter correctement ce mode de fonctionnement du μP (appelé mode « LOAD ») doit faire l'effort d'analyser les chronogrammes correspondants donnés figure 22. Commentons ces diagrammes :

1. t_1 est défini comme l'époque à laquelle les opérations suivantes sont supposées exécutées :

a) $\overline{\text{WAIT}}$ étant à 1, $\overline{\text{CLEAR}}$ est mis à 0.

b) Puis $\overline{\text{WAIT}}$ est mis à 0. Le μP décrit alors un cycle d'initialisation qui se termine à l'instant t_1 . Tant que $\overline{\text{DMA}} - \overline{\text{IN}}$ est à 1, le μP est installé dans un état d'attente pendant lequel la mémoire est en HI. Par ailleurs le contenu de R (0) est nul.

2. Si $\overline{\text{DMA}} - \overline{\text{IN}}$ est mis à 0, le μP décrit un cycle DMA (cycle d'écriture) débutant à l'instant t_2 et pendant lequel le mot W_0 présent sur les clés D_7 à D_0 est enregistré dans la mémoire 00. $\overline{\text{DMA}} - \overline{\text{IN}}$ est remis à 1 par SC 1 dès le début de ce cycle.

3. Dès la fin du cycle précédent, c'est-à-dire à l'instant t_3 , démarre un *cycle de lecture* pendant lequel le mot M (00) (normalement égal à W_0) contenu dans la mémoire à l'adresse 00 est placé sur le bus des données. Comme nous le verrons plus loin, le

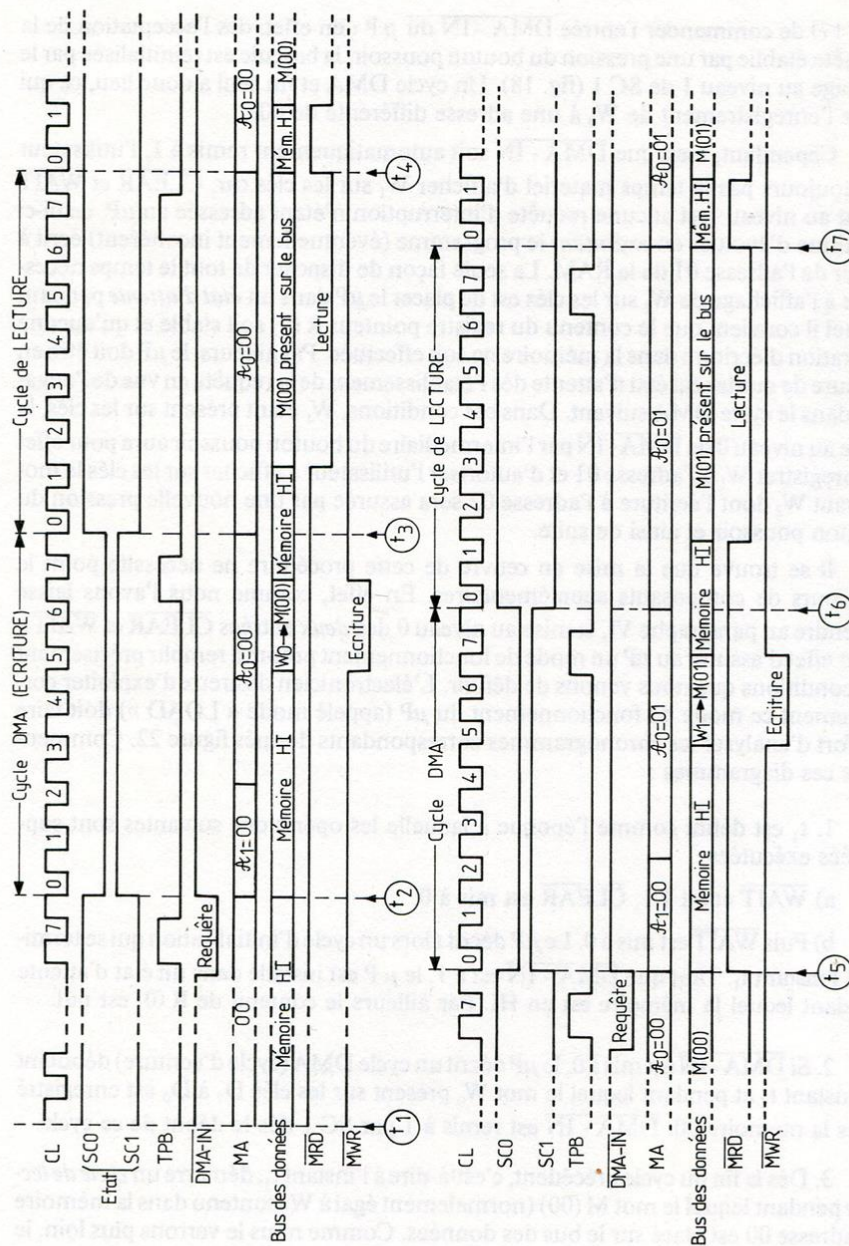


Fig. 22. Chronogrammes du mode LOAD

coupleur de la maquette peut être chargé de transférer ce mot vers un couple d'afficheurs hexadécimaux.

4. A l'instant t_4 correspondant à la fin du cycle de lecture, le μP se place dans un état d'attente pendant lequel se répète indéfiniment le cycle de lecture précédent. L'utilisateur a donc tout le temps d'afficher sur les clés le mot suivant W_1 .

5. Si $\overline{DMA - IN}$ est de nouveau mis à 0, le μP décrit un cycle DMA débutant à l'instant t_5 et pendant lequel le mot W_1 présent sur les clés est mémorisé à l'adresse suivante 01.

6. Dès la fin du cycle précédent a lieu un cycle de lecture pendant lequel le mot $M(01)$ est mis sur le bus des données ; ce cycle de lecture se répète aussi longtemps qu'une nouvelle requête de DMA n'est pas établie, ce qui permet à l'utilisateur de placer sur les clés le mot suivant W_2 en vue de son chargement à l'adresse 02, et ainsi de suite.

Remarques

1. Lorsque vous désirez faire exécuter par le μP le programme que vous venez d'écrire dans la RAM, il faut réinitialiser le système en plaçant \overline{WAIT} à 1 puis \overline{CLEAR} à 1.

2. Dans le mode LOAD, la sortie \overline{SR} de la bascule JK doit être reliée à l'entrée $\overline{DMA - IN}$. Or quand vous vous apprêtez à mettre le μP dans le mode LOAD en vue de charger votre programme, il n'est pas sûr que $\overline{S.R.}$ soit au niveau 1. Supposez que $\overline{S.R.}$ soit initialement à 0 : dès que \overline{CLEAR} et \overline{WAIT} sont mis à 0, le μP décrit un cycle DMA pendant lequel le mot présent sur les clés est enregistré dans la mémoire à l'adresse 00 ; si ce mot n'est pas W_0 il faut réinitialiser le système en mettant \overline{WAIT} à 1 puis à 0. Vous remarquez que maintenant la sortie de la bascule est à 1 : le μP attend votre premier mot W_0 .

VIII. 2. La visualisation du contenu de la mémoire

La maquette A contient tous les composants nécessaires à la mise en œuvre de la procédure précédemment décrite. Cependant, l'existence de cycles de lecture à l'intérieur de ce mode de fonctionnement suggère qu'il doit être possible non seulement de charger rapidement un programme dans la mémoire mais d'en visualiser le contenu.

A cet effet, constatons que le coupleur CDP 1852 de la maquette B est en mesure de transmettre une donnée sur son bus de sortie DO si et seulement si son déclenchement est autorisé par la mise au niveau 1 de son entrée CS. Or c'est actuellement la sortie N_0 du μP qui est reliée à CS ; par conséquent, si nous désirons com-

muniquer au monde extérieur les informations circulant sur le bus des données lors du fonctionnement du μP dans le mode LOAD, il est nécessaire que CS réponde à l'équation logique suivante :

$CS = N_0 \vee L$, L étant un signal qui doit être au niveau 1 si le μP fonctionne dans le mode LOAD. Or ce mode est établi pour la combinaison (0,0) sur les entrées de commande (\overline{CLEAR} , \overline{WAIT}) et par suite :

$$L = \overline{CLEAR} \vee \overline{WAIT}$$

Remarquons cependant qu'il n'est pas gênant de valider le coupleur pour la combinaison (0,1) appliquée sur les entrées (\overline{CLEAR} , \overline{WAIT}) puisqu'alors le μP est bloqué ; il s'ensuit qu'il est suffisant que $L = \overline{CLEAR}$, ce qui donne :

$$CS = N_0 \vee \overline{CLEAR} = \overline{N_0} \cdot \overline{CLEAR} \text{ (voir fig. 23).}$$

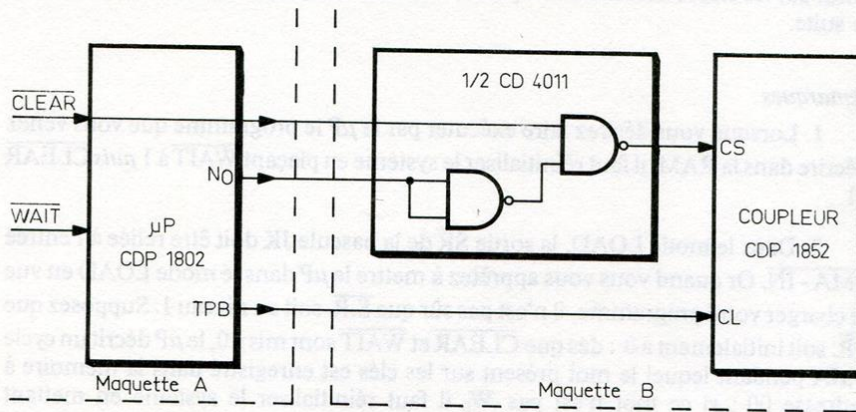


Fig. 23

Ainsi, lors du fonctionnement du μP dans le mode LOAD, tout mot W présent sur les clés est transmis sur le bus de sortie DO du coupleur immédiatement après son écriture dans la mémoire. W est donc affiché en binaire sur les LED DO_7 à DO_0 . Si l'utilisateur le désire, il peut disposer d'une lecture hexadécimale du mot W en insérant deux afficheurs TIL 311 à l'endroit prévu à cet effet sur la maquette B.

En réalité il n'y a aucun avantage à visualiser le mot W immédiatement après son enregistrement dans la RAM. En effet, il est de toutes façons nécessaire de l'« afficher » sur les clés *avant* sa mémorisation. Notre montage paraîtra cependant digne d'intérêt après considération de la remarque suivante :

Supposez que la broche \overline{MWR} de la mémoire soit constamment imposée au niveau 1, celle-ci ne peut donc être que lue. Il en résulte que, lors du fonctionnement du

μP dans le mode LOAD, seuls les cycles de lecture seront autorisés. Par conséquent, le système étant initialisé, une pression du bouton poussoir aura pour effet non pas l'écriture du mot W_0 à l'adresse 00 puis sa visualisation, mais *seulement* la visualisation sur les afficheurs du mot M (00) contenu dans la mémoire à l'adresse 00.

Une nouvelle pression du bouton poussoir conduira à la visualisation de M (01) ; une pression suivante aboutira à la visualisation de M (02) et ainsi de suite.

S'il est interdit à la mémoire d'être écrite, elle est fonctionnellement équivalente à une ROM. Ainsi, si l'inverseur de la figure 24 est dans la position « ROM », les signaux envoyés en vue d'écrire la mémoire sont sans effet ; notez que la résistance R évite à la sortie \overline{MWR} d'être en court-circuit lors d'un cycle d'écriture déroulé par le μP . L'utilisateur est donc en mesure de lire un programme qu'il vient de charger soit pour le vérifier, soit pour le rectifier. Dans ce dernier cas, \mathcal{A} étant l'adresse de la ligne mémoire contenant le mot à modifier, il est nécessaire de présenter sur les clés D_7 à D_0 le mot correct W lors de l'apparition de $\mathcal{A} - 01$ sur le bus des adresses ; on place alors l'inverseur en position « RAM » et on demande au μP d'écrire W à l'adresse \mathcal{A} par une pression du bouton poussoir.

Les schémas synoptiques complets de la maquette A et de la maquette B sont donnés respectivement figures 25 et 26.

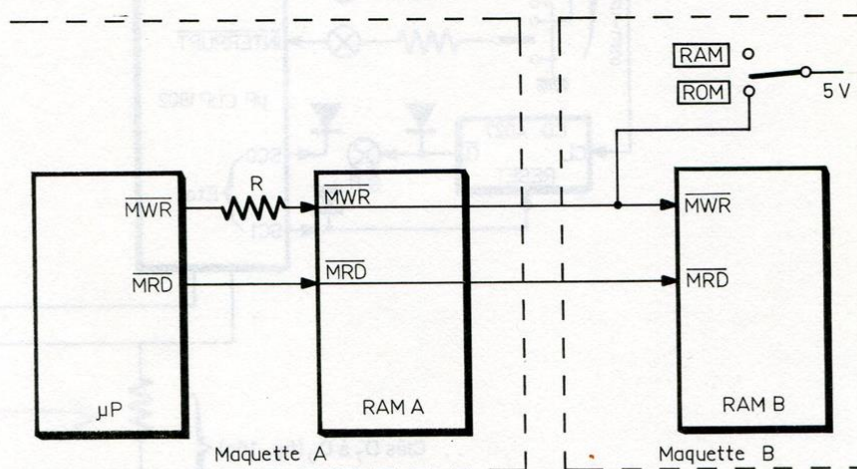


Fig. 24

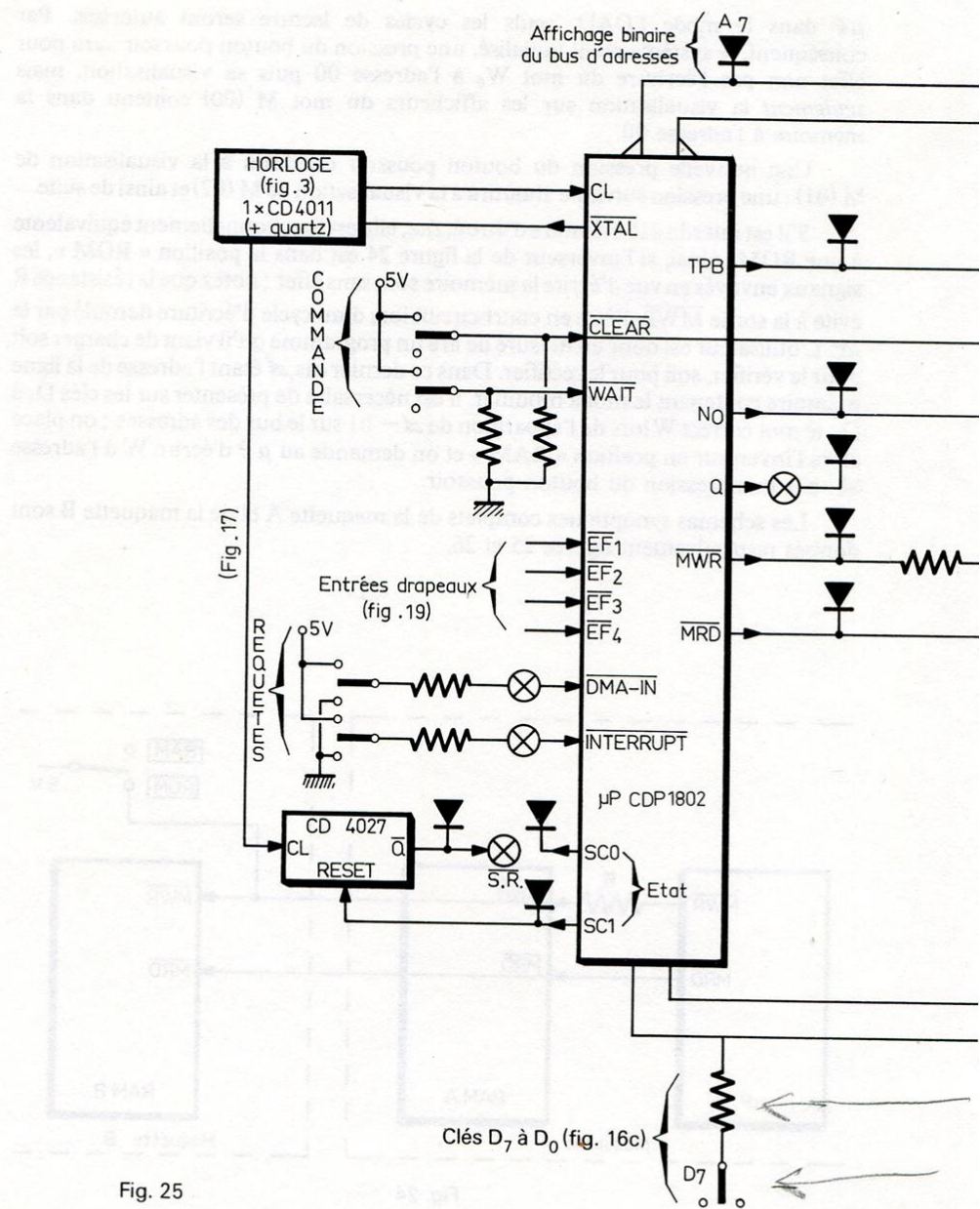
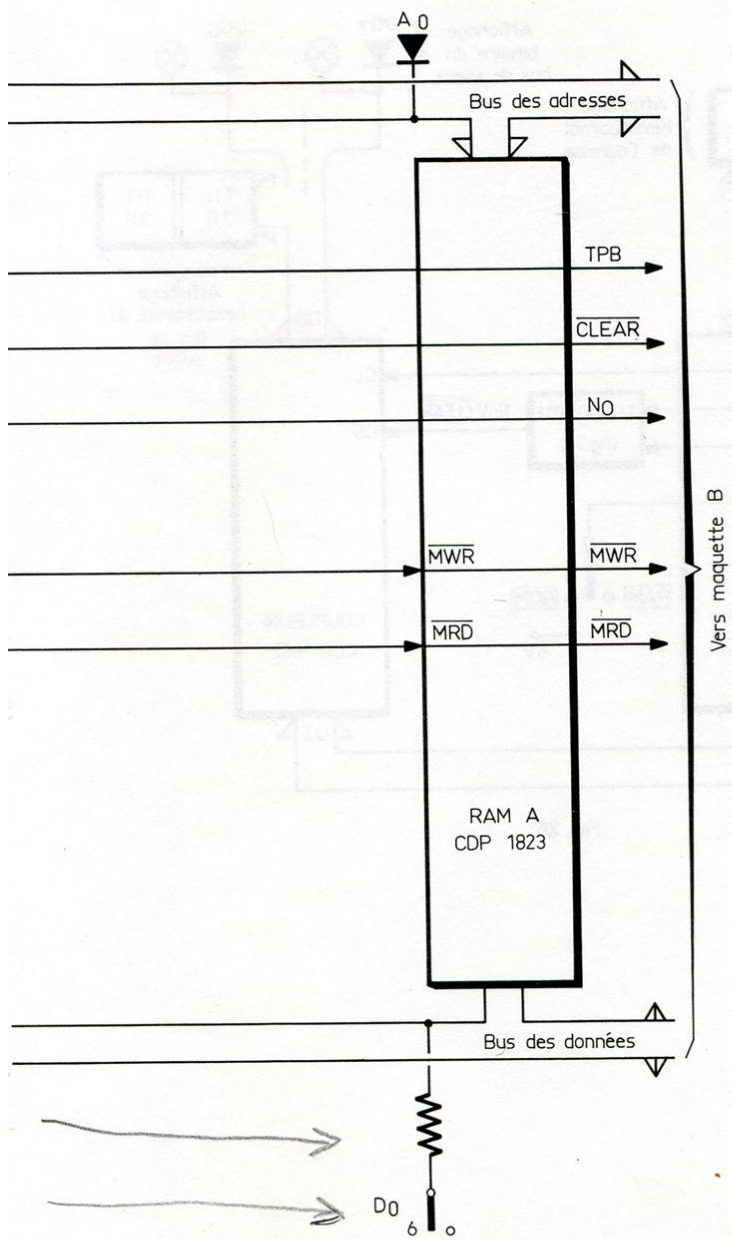


Fig. 25



Chapitre 8

RÉALISATION PRATIQUE DE LA MAQUETTE A

Nous proposons de guider le lecteur dans la réalisation pratique de la maquette A dont le schéma de câblage complet est donné figure 1.

La figure 2 représente le dessin du circuit imprimé côté cuivre (1).

La figure 3 représente le schéma d'implantation des composants et donne l'aspect de la maquette A terminée. La figure 3-bis indique ce que veulent dire les symboles utilisés.

La figure 4 indique de quelle façon il faut effectuer le câblage des commandes, c'est-à-dire des inverseurs et du bouton poussoir.

I. Présentation du matériel utilisé

Avant de se lancer dans la construction, il faut se reporter à la figure 3. Nous allons décrire brièvement l'organisation générale de la maquette A, tout en rappelant le rôle des composants essentiels.

1. Le mot de 8 bits supporté par le bus des adresses est visualisé par 8 LED TIL 209 ou équivalent, rangées horizontalement et notées de A_7 à A_0 . La commande de ces LED est assurée par 8 transistors 2N2222 (ou équ.) auxquels sont associées 8 résistances de base de $33\text{ k}\Omega$ et 8 résistances de limitation de courant de 220Ω .

2. Les signaux de lecture et d'écriture \overline{MRD} et \overline{MWR} , le signal de synchronisation TPB, le signal N_0 de validation d'un coupleur entrée/sortie éventuel, et la sortie Q, sont visualisés par 5 LED rangées verticalement entre le microprocesseur et la mémoire. Les résistances de base des transistors commandant ces LED sont également de $33\text{ k}\Omega$, sauf celle concernant la sortie Q qui est de $5,6\text{ k}\Omega$. La douille associée à la sortie Q est reliée au collecteur du transistor correspondant.

3. Les signaux d'état SC0 et SC1 sont visualisés par 2 LED rangées verticalement entre le microprocesseur et le boîtier CD4011.

4. La sortie SC1 est reliée à l'entrée de remise à zéro de la bascule JK CD4027. La sortie \overline{Q} de cette bascule est notée $\overline{S.R.}$ (Requête de Service) et est visualisée par

(1) Les lecteurs que la gravure du circuit imprimé rebuterait recevront l'adresse d'un fournisseur s'ils nous envoient une enveloppe timbrée à leurs nom et adresse.



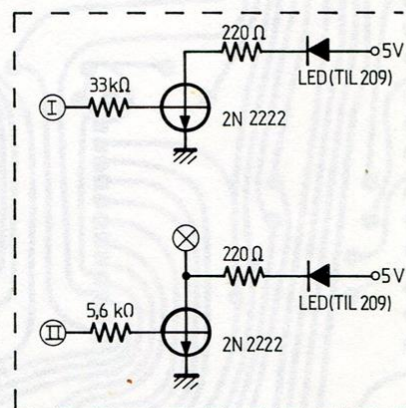
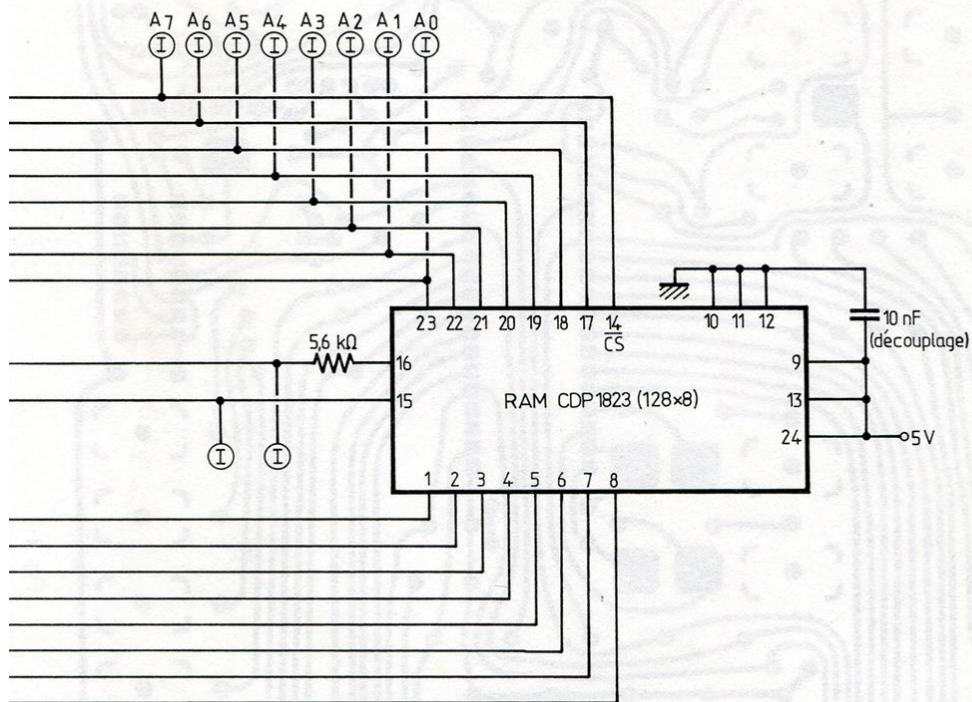
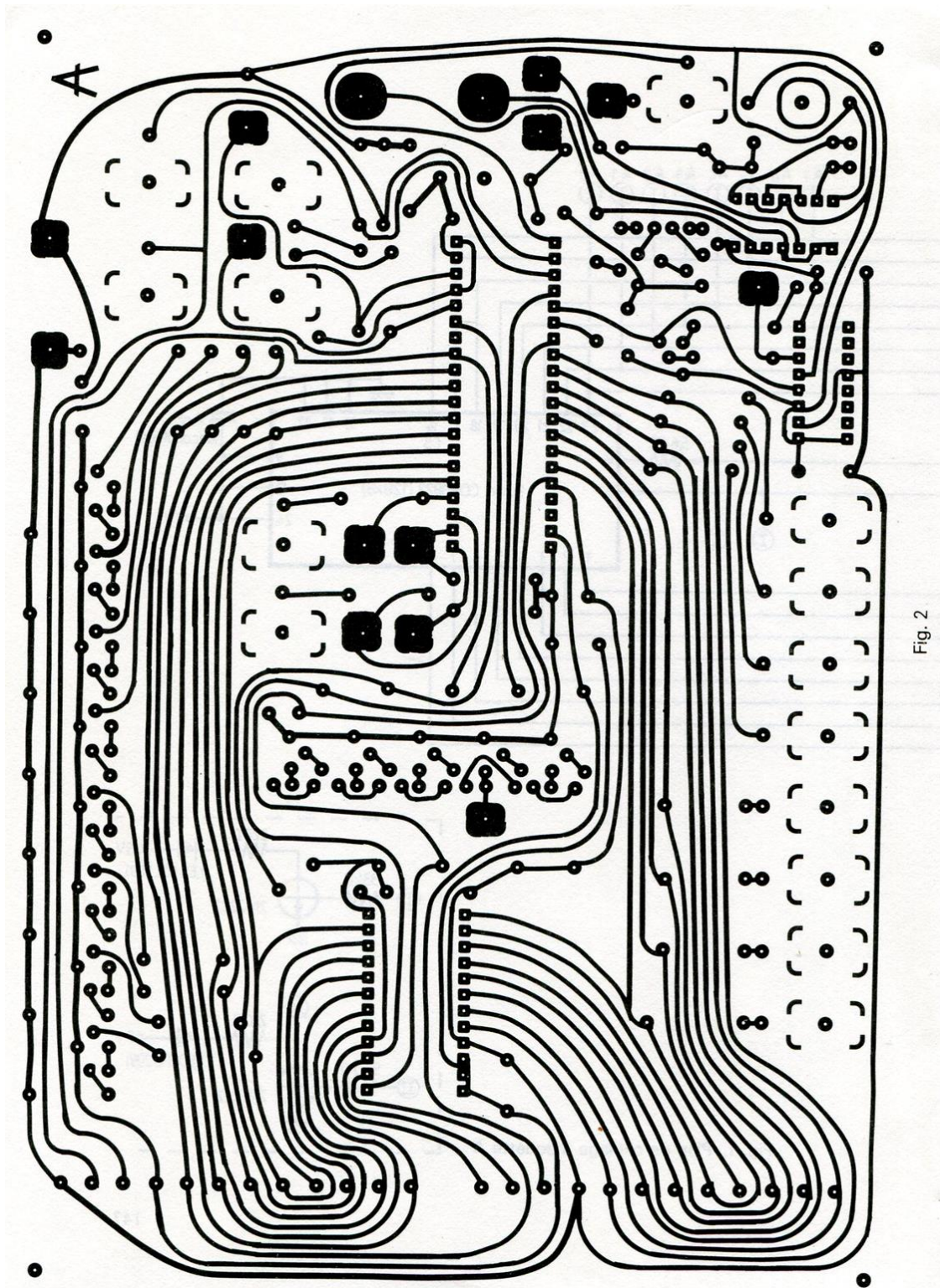


Fig. 1. Plan de câblage maquette A



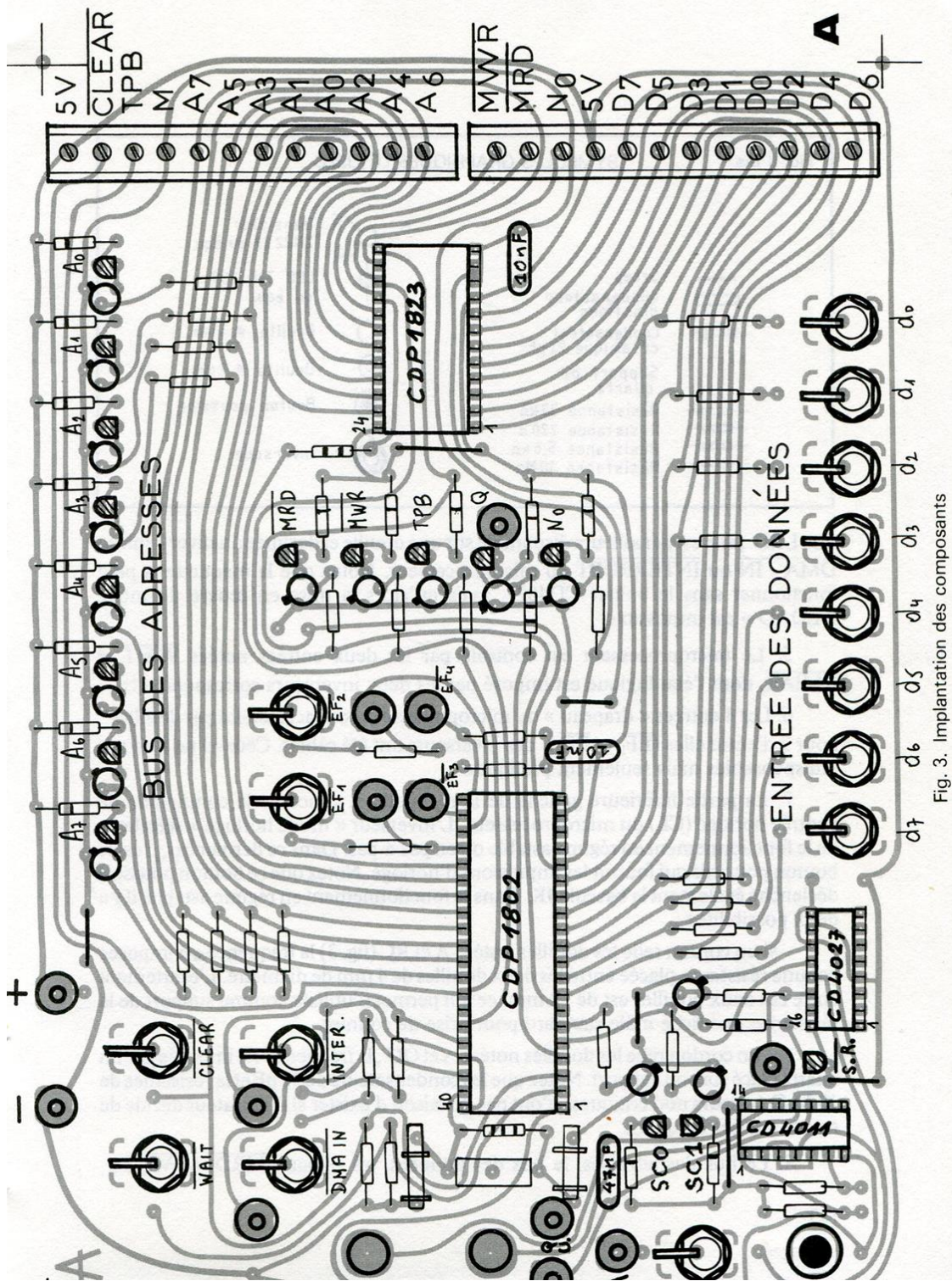
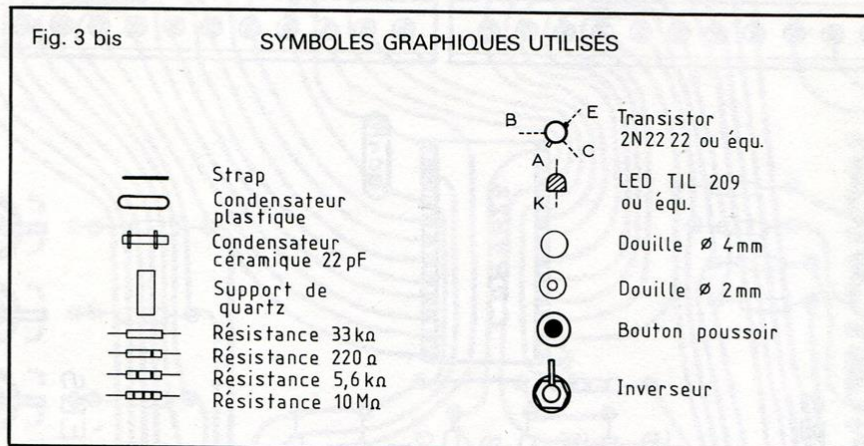


Fig. 3. Implantation des composants

Fig. 3 bis

SYMBÔLES GRAPHIQUES UTILISÉS



une LED ; elle est par ailleurs disponible sur une douille et destinée à activer l'entrée $\overline{\text{DMA}} - \text{IN}$ ou $\overline{\text{INTERRUPT}}$ du microprocesseur. Notez que la maquette A peut fonctionner sans le boîtier CD4027, mais qu'alors la mise en œuvre du mode « LOAD » est impossible.

5. Le microprocesseur est contrôlé par les deux entrées notées $\overline{\text{WAIT}}$ et $\overline{\text{CLEAR}}$, dont l'état logique est imposé par les deux inverseurs correspondants.

6. Les 4 entrées « drapeau » du microprocesseur sont accessibles par douilles ; pour 2 d'entre elles ($\overline{\text{EF}}_1$ et $\overline{\text{EF}}_2$) des inverseurs ont été câblés. Ceux-ci ne sont pas indispensables mais seulement pratiques.

7. La partie inférieure gauche de la maquette A concerne la commande de l'entrée horloge (CL) du microprocesseur. L'inverseur « mode horloge » sélectionne le fonctionnement en régime astable ou en pas-à-pas. Dans ce dernier cas, c'est le bouton poussoir qui fournit les impulsions d'horloge. Notez que ce bouton poussoir déclenche également la bascule JK. Dans le fonctionnement en régime astable, il y a deux possibilités :

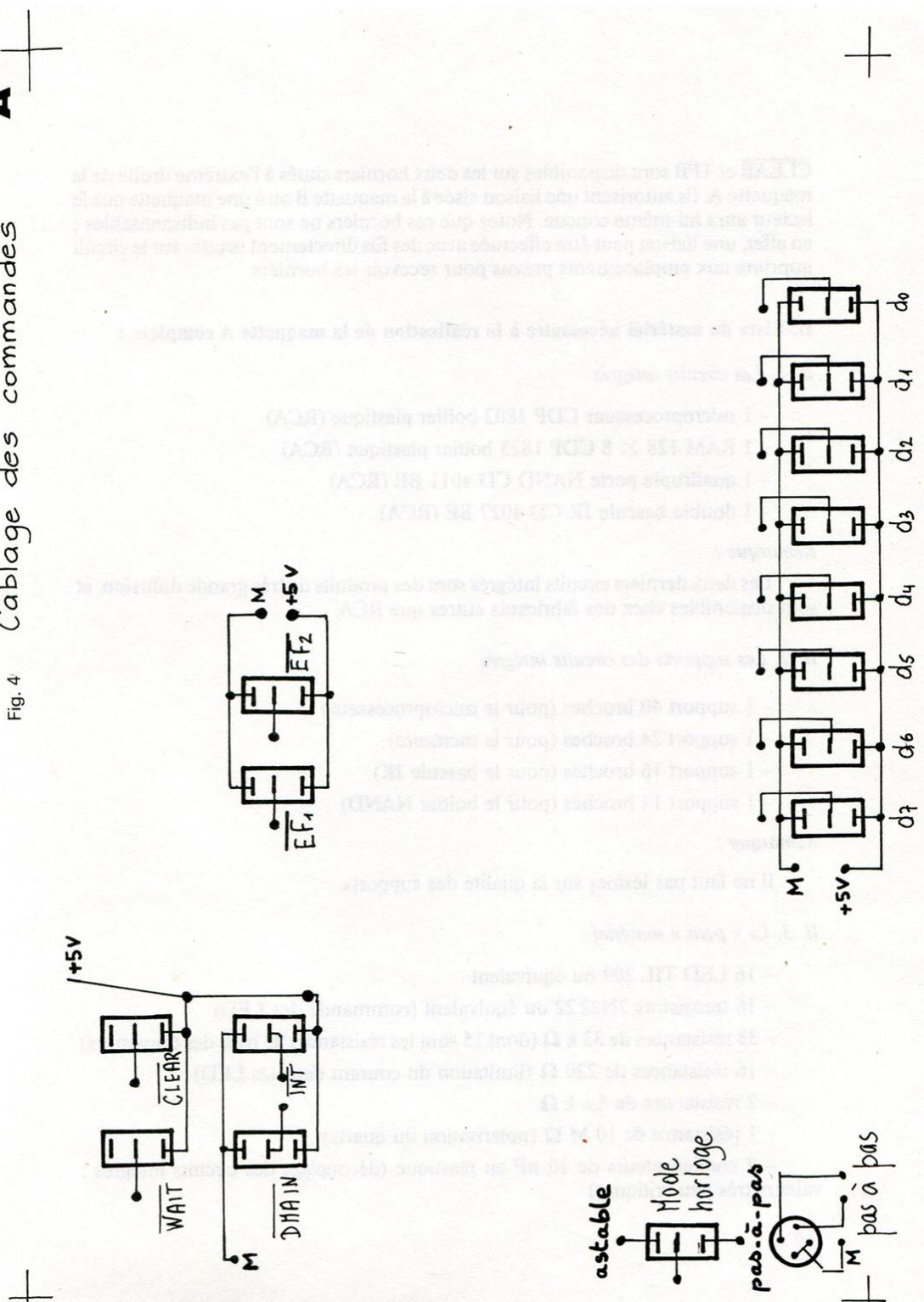
– si un cordon relie les douilles notées A et RC (fig. 3) la fréquence est imposée par une résistance placée entre les deux douilles de 4 mm de diamètre. L'écartement entre ces deux douilles est de 19 mm, ce qui permet d'utiliser comme support de la résistance une fiche mâle standard pour prise de courant.

– si un cordon relie les douilles notées A et QU., la fréquence est imposée par un quartz placé sur son support. Notez que les condensateurs de 22 pF et la résistance de 10 M Ω (valeurs non critiques) n'ont pas de raison d'exister si l'utilisateur décide de ne pas utiliser de quartz.

8. Le bus des adresses, le bus des données, les signaux $\overline{\text{MRD}}$, $\overline{\text{MWR}}$, N_0 ,

Fig. 4 Câblage des commandes

A



CLEAR et TPB sont disponibles sur les deux borniers situés à l'extrême droite de la maquette A. Ils autorisent une liaison aisée à la maquette B ou à une maquette que le lecteur aura lui-même conçue. Notez que ces borniers ne sont pas indispensables ; en effet, une liaison peut être effectuée avec des fils directement soudés sur le circuit imprimé aux emplacements prévus pour recevoir les borniers.

II. Liste du matériel nécessaire à la réalisation de la maquette A complète :

II. 1. Les circuits intégrés

- 1 microprocesseur CDP 1802 boîtier plastique (RCA)
- 1 RAM 128×8 CDP 1823 boîtier plastique (RCA)
- 1 quadruple porte NAND CD 4011 BE (RCA)
- 1 double bascule JK CD 4027 BE (RCA)

Remarque :

Les deux derniers circuits intégrés sont des produits de très grande diffusion, et sont disponibles chez des fabricants autres que RCA.

II. 2. Les supports des circuits intégrés

- 1 support 40 broches (pour le microprocesseur)
- 1 support 24 broches (pour la mémoire)
- 1 support 16 broches (pour la bascule JK)
- 1 support 14 broches (pour le boîtier NAND)

Remarque :

Il ne faut pas lésiner sur la qualité des supports.

II. 3. Le « petit » matériel :

- 16 LED TIL 209 ou équivalent
- 16 transistors 2N2222 ou équivalent (commande des LED)
- 33 résistances de $33 \text{ k } \Omega$ (dont 15 sont les résistances de base des transistors)
- 16 résistances de $220 \text{ } \Omega$ (limitation du courant dans les LED)
- 2 résistances de $5,6 \text{ k } \Omega$
- 1 résistance de $10 \text{ M } \Omega$ (polarisation du quartz)
- 2 condensateurs de 10 nF en plastique (découplage des circuits intégrés : valeurs très peu critiques)

- 1 condensateur de 47 nF en plastique (contrôle de la fréquence de l'oscillateur RC)
- 2 condensateurs céramiques de 22 pF (circuit résonnant du quartz)
- 1 support de quartz

Remarque :

La valeur des composants n'est pas critique. On pourra prendre pour les résistances des types 1/4 W, 10 %.

II. 4. Les commandes

– 15 inverseurs miniatures (8 pour l'entrée des données, 2 pour le contrôle du microprocesseur, 2 pour les requêtes d'interruption, 2 pour les entrées \overline{EF}_1 et \overline{EF}_2 , et 1 pour la sélection du mode horloge). Ces inverseurs sont miniatures dans le sens où ils doivent pouvoir être insérés aux emplacements prévus à cet effet ! (voir le circuit imprimé figure 2). On peut par exemple prendre des inverseurs DJET fabriqués par SECME sous la référence 3 125 201 21.

– 1 bouton poussoir miniature. (type inverseur). On peut prendre par exemple un bouton poussoir DJET fabriqué par SECME sous la référence 3 525 511 01.

III. 5. Les douilles et les borniers

– 13 douilles non isolées avec écrou, de diamètre intérieur 2 mm.
Par exemple, chez SECME, la référence de la douille avec son écrou est 52 302 00 47.

– 2 douilles non isolées avec écrou, de diamètre intérieur 4 mm (supportent la résistance qui fixe la fréquence de l'oscillateur RC)

– 2 borniers à 12 bornes.

La référence est KRE 12 chez Radio-Relais.

Remarque

Ce sont les écrous qui assurent la liaison électrique entre les douilles et la piste du circuit imprimé

III. Réalisation pratique de la maquette A

Nous partons du principe que le lecteur dispose du circuit imprimé gravé exactement selon le dessin de la figure 2 (voir note page 145). Le diamètre des trous de perçage doit être le suivant :

- \varnothing 0,9 mm pour les supports de circuits intégrés, les transistors, les LED, les résistances, les condensateurs.

- \varnothing 4,4 mm pour les douilles de 2 mm (valeur à ne pas dépasser pour ne pas « manger » tout le cuivre)
- \varnothing 4,9 mm pour les douilles de 4 mm
- \varnothing 2,5 mm pour le support de quartz
- \varnothing 1,2 mm pour les borniers (valeur à ne pas dépasser)
- \varnothing 6,3 mm pour les inverseurs et le bouton poussoir

Remarque : Cette dernière valeur peut être sujette à de légères variations, ceci dépendant du type d'inverseur utilisé.

L'adoption de la procédure de montage que nous allons développer, garantit à coup sûr le fonctionnement de la maquette A. Nous conseillons donc de respecter les étapes suivantes :

- 1) Souder les 4 supports de circuits intégrés.
- 2) Placer les 2 douilles pour l'alimentation 5V. Les contacts devant toujours être excellents, souder chacun des 2 écrous sur la piste de cuivre.
- 3) Placer les straps (il y a 6 straps).
- 4) Souder toutes les LED ainsi que tous les transistors en respectant leur orientation ; souder également les 16 résistances de base associées (15 de 33 k Ω et 1 de 5,6 k Ω) ainsi que les 16 résistances de limitation de courant de 220 Ω .
- 5) La maquette étant alimentée (une pile de 4,5 V est ici suffisante), établir au moyen d'un cordon une liaison entre la douille + et chacun des 16 points B' (fig. 5). La LED correspondante doit évidemment s'allumer ; si ce n'est pas le cas, voyez ce qui ne va pas avant de poursuivre.
- 6) Reprendre le 5) en prenant les points B' directement sur le support du micro-processeur. Les numéros des broches du support à tester sont les suivants :
 - n^{os} 32 à 25 pour les LED notées de A₇ à A₀ (Adresses).
 - n^{os} 5 et 6 pour les LED SC1 et SCO
 - n^{os} 7, 35, 33, 4, 19 pour les LED visualisant respectivement les signaux $\overline{\text{MRD}}$, $\overline{\text{MWR}}$, TPB, Q, N₀.
- 7) Souder toutes les résistances et condensateurs.
- 8) Placer et câbler tous les inverseurs (fig. 4). Ce câblage doit être testé en utilisant l'un des transistors de la maquette, comme l'indique la figure 6. Pour chacun des inverseurs, sauf celui qui sélectionne le mode horloge, le résultat du test est indiqué figure 6 bis.
- 9) Câbler le bouton poussoir. Insérer le boîtier CD 4011 sur son support et mettre sous tension la maquette. L'inverseur de sélection du mode horloge étant en position « pas-à-pas », une pression du bouton poussoir engendre un niveau 1 sur la

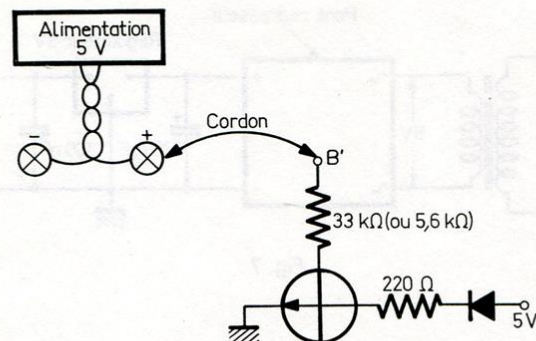


Fig. 5

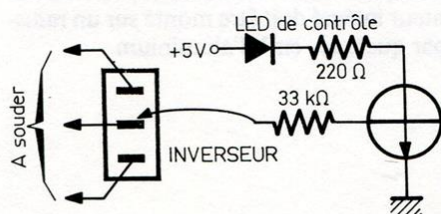


Fig. 6

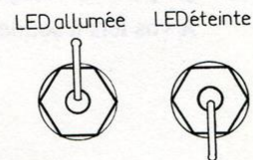


Fig. 6 bis

patte n°1 du support du microprocesseur. Le vérifier en utilisant comme précédemment un transistor et sa LED associée comme sonde de test. Si ce n'est pas le cas, voyez ce qui ne va pas avant de poursuivre.

10) Placer les douilles de 2 mm notées A, RC et QU., ainsi que les douilles de 4 mm. Le boîtier CD 4011 étant toujours le seul circuit inséré sur son support, choisir une résistance de l'ordre du mégohm que l'on mettra entre les deux douilles de 4 mm. L'inverseur de sélection du mode horloge étant en position « astable », et un cordon reliant les douilles A et RC, une oscillation très basse fréquence apparaît sur la patte n°1 du support du microprocesseur.

11) Placer les douilles restantes, et éventuellement le support du quartz et les borniers.

12) Insérer les circuits intégrés sur leur support : la maquette A est prête à fonctionner.

Signalons pour terminer que cette maquette est destinée à fonctionner avec une alimentation de + 5V. Il ne faut en aucun cas dépasser 6V ; le fonctionnement correct n'est plus garanti en dessous de 4V.

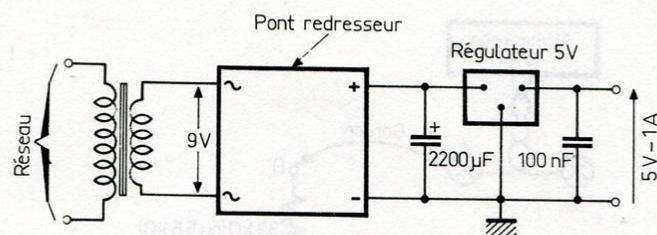


Fig. 7

La consommation dépend essentiellement du nombre de LED allumées et ne peut excéder 200 m A ; il vaut mieux prévoir cependant une alimentation 5V - 1A dont le schéma est donné figure 7 à titre indicatif : le secondaire du transformateur doit pouvoir fournir 1A sous 9V ; le régulateur intégré doit être monté sur un radiateur qui peut être simplement constitué par quelques cm² d'aluminium.

A vos fers à souder, donc !

Chapitre 9

RÉALISATION PRATIQUE DE LA MAQUETTE B

Le plan de câblage de la maquette B, complète, est donné figure 1.

La figure 2 représente le dessin du circuit imprimé côté cuivre (1).

La figure 3 représente le schéma d'implantation des composants et donne l'aspect de la maquette B terminée

I. Présentation et rappel du rôle du matériel utilisé

Le lecteur est prié de se reporter à la figure 3.

1. Le mot de 8 bits supporté par le bus des adresses issu de la maquette A est visualisé par 2 afficheurs hexadécimaux TIL 311. Par ailleurs, la RAM supplémentaire CDP 1823 porte la capacité mémoire de l'ensemble maquette A — maquette B à 256 octets.

2. Le registre de sortie CDP 1852 a pour fonction de mémoriser une donnée présente sur le bus des données, soit lors de l'exécution d'une instruction de sortie qui active la sortie N_0 du microprocesseur, soit lors du fonctionnement dans le mode LOAD. C'est le rôle du boîtier CD 4011 de valider le registre de sortie dans l'un ou l'autre de ces deux cas.

3. La donnée mémorisée par le registre de sortie est visualisée par 2 afficheurs hexadécimaux TIL 311, et par les LED notées de DO_7 à DO_0 . Les collecteurs des transistors de commande de ces LED sont munis de douilles, autorisant ainsi la liaison à divers organes électroniques susceptibles d'être commandés par le microprocesseur.

4. Dans le mode de fonctionnement LOAD, l'inverseur ROM - RAM placé en position « ROM » interdit à la mémoire d'être écrite. Ceci permet de lire rapidement le contenu de la mémoire et de ne pas perdre de temps dans la mise au point d'un programme.



Remarque

Le registre CDP 1852 fonctionne seulement si le boîtier CD4011 est inséré sur son support. Par ailleurs, les afficheurs hexadécimaux et le boîtier mémoire ne sont pas indispensables au bon fonctionnement de la maquette B ; le lecteur se rendra cependant rapidement compte de leur commodité.

(1) Voir note page 145.

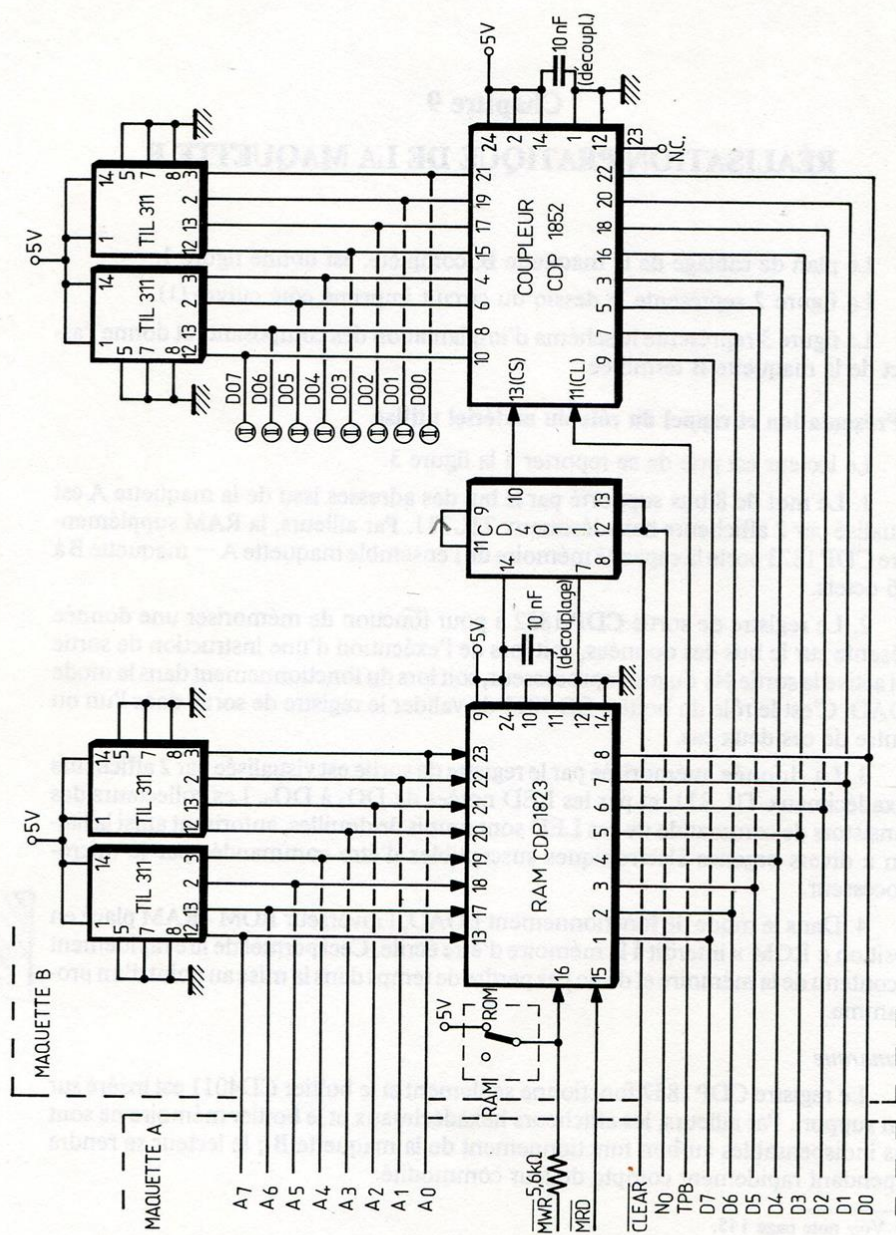
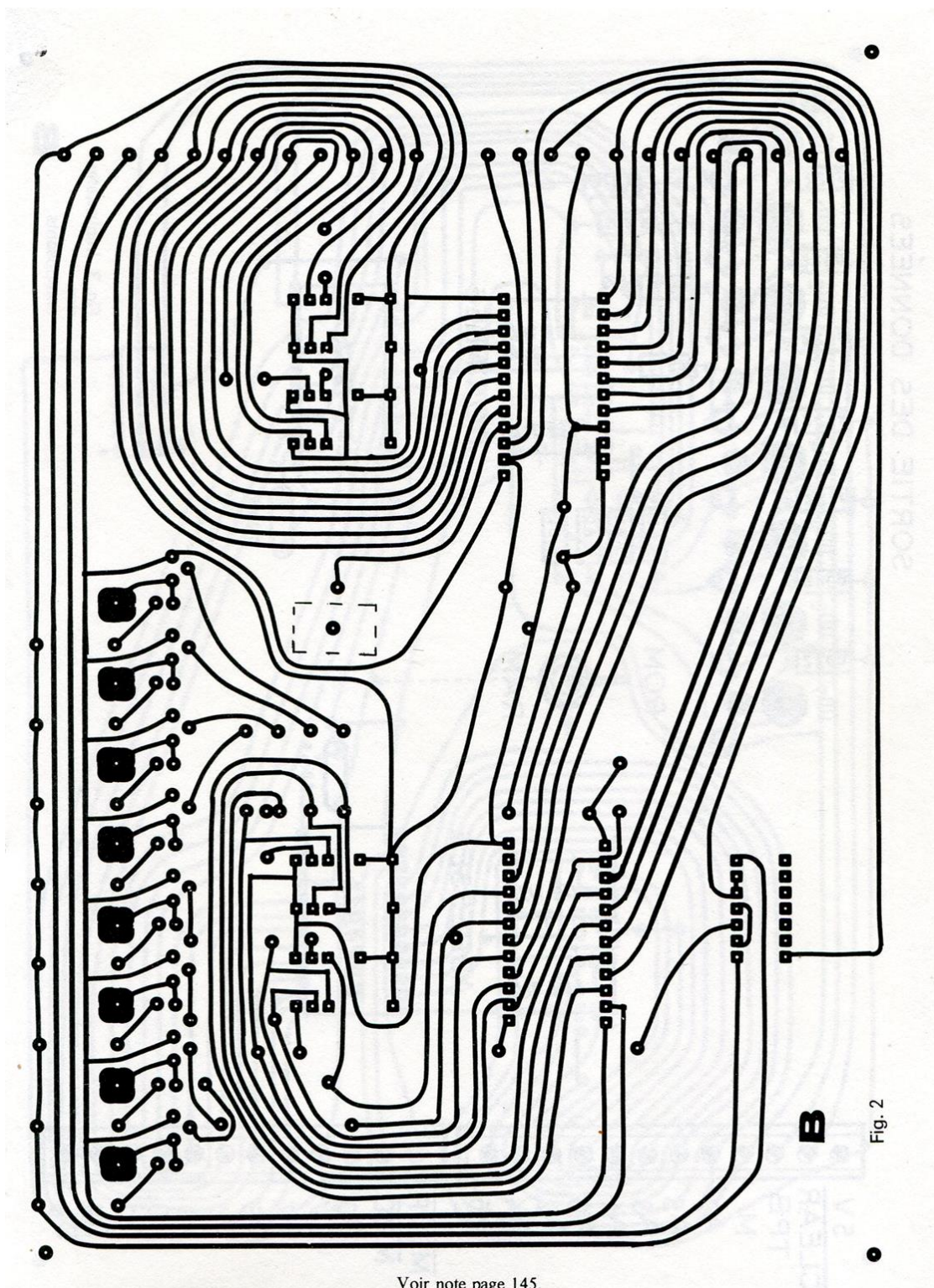


Fig. 1. Plan de câblage de la maquette B



B

Fig. 2

Voir note page 145.

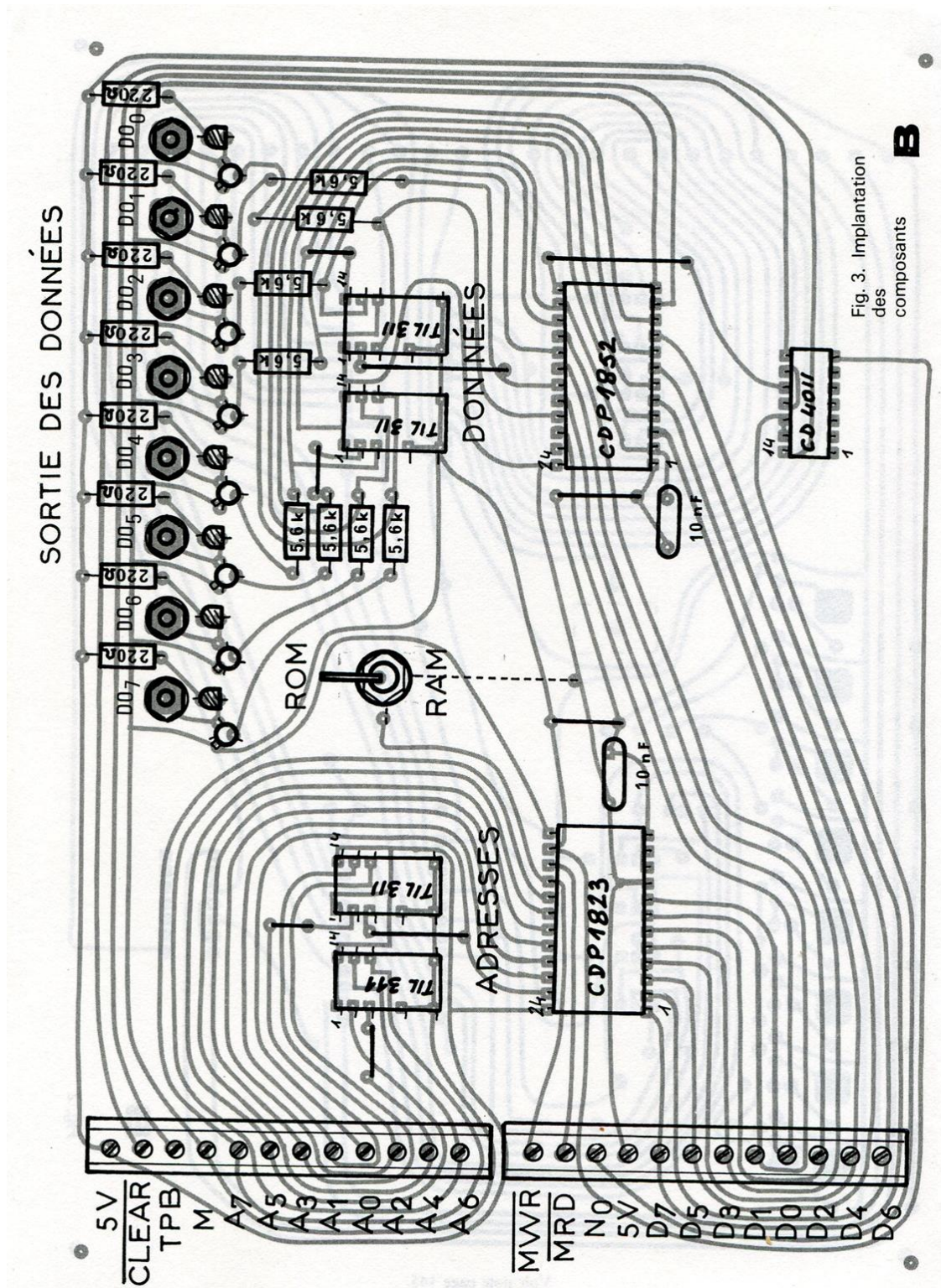


Fig. 3. Implantation
des
composants

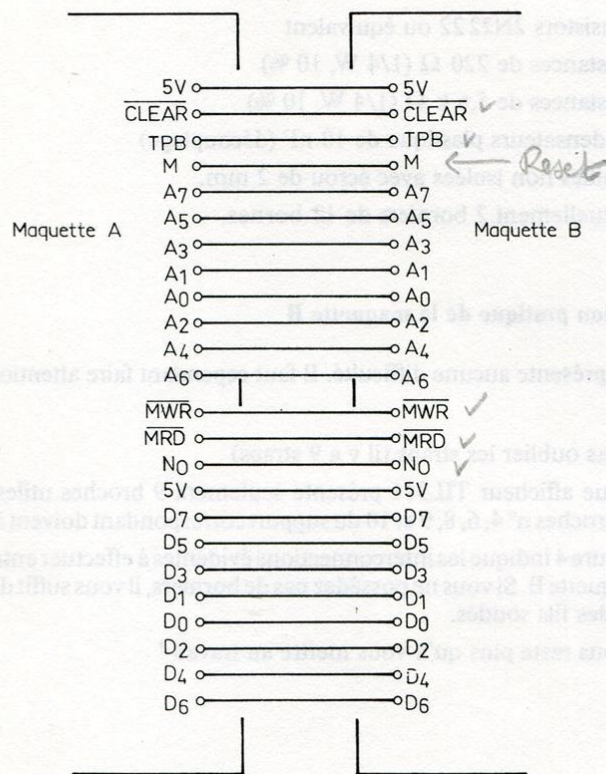


Fig. 4

II. Liste du matériel nécessaire à la réalisation de la maquette B

- 1 RAM 128×8 CDP 1823 boîtier plastique (RCA)
- 1 registre de sortie CDP 1852 boîtier plastique (RCA)
- 1 quadruple porte NAND CD 40 11 BE (RCA)
- 2 supports de 24 broches
- 5 supports de 14 broches (dont 4 pour les afficheurs)
- 1 inverseur miniature
- 8 LED TIL 209 ou équivalent
- 4 afficheurs hexadécimaux TIL 311

- 8 transistors 2N22 22 ou équivalent
- 8 résistances de 220 Ω (1/4 W, 10 %)
- 8 résistances de 5,6 k Ω (1/4 W, 10 %)
- 2 condensateurs plastique de 10 nF (découplage)
- 8 douilles non isolées avec écrou de 2 mm.
- Eventuellement 2 borniers de 12 bornes.

III. Réalisation pratique de la maquette B

Elle ne présente aucune difficulté. Il faut cependant faire attention aux points suivants :

1. Ne pas oublier les straps (il y a 9 straps)
2. Chaque afficheur TIL311 présente seulement 9 broches utiles. Par conséquent, les 5 broches n° 4, 6, 8, 9 et 10 du support correspondant doivent être coupées.
3. La figure 4 indique les interconnexions évidentes à effectuer entre la maquette A et la maquette B. Si vous ne possédez pas de borniers, il vous suffit d'effectuer les liaisons par des fils soudés.

Il ne vous reste plus qu'à vous mettre au travail !

Troisième partie

ÉTUDE DES PRINCIPES FONDAMENTAUX DE LA MISE EN ŒUVRE ET DE LA PROGRAMMATION DU MICROPROCESSEUR

Chapitre 10

ÉTUDE EN « PAS-A-PAS » DE L'ENREGISTREMENT ET DU DÉROULEMENT D'UN PROGRAMME ÉLÉMENTAIRE

I. Position du problème

Voici notre projet : nous désirons analyser de façon détaillée la procédure de mise en œuvre du μP en vue de lui faire exécuter un programme répondant à un but déterminé. Le μP présente une sortie Q commandant une LED sur la maquette ; nous nous proposons de demander au μP d'allumer cette LED. Il est bien évident que, d'un point de vue pratique, il n'est pas nécessaire d'utiliser un μP pour allumer une lampe ! Par ailleurs, il n'est pas non plus nécessaire d'imaginer une tâche compliquée pour être en mesure de comprendre les différents processus élaborés par le μP lors de son fonctionnement.

Le μP est un composant électronique capable d'effectuer automatiquement un travail déterminé à la seule condition qu'on lui fournisse toutes les instructions utiles à cet effet ; c'est le rôle de la mémoire de contenir ces instructions. Par conséquent, notre désir de mettre la sortie Q au niveau 1 (en vue d'allumer la LED) doit être préalablement communiqué au μP , et ceci dans le seul langage qu'il comprenne : le langage binaire (langage machine). L'utilisateur se rend ainsi compte qu'il doit commencer par effectuer lui-même le travail consistant à traduire le projet « mettre Q à 1 » en langage machine. Or le jeu d'instructions du μP est précisément le dictionnaire qui permet cette traduction ; à l'instruction SET Q (mnémonique SEQ) correspond le code 7 B que le constructeur a jugé commode de donner en notation hexadécimale. 7 B (c'est-à-dire 0111 1011 en binaire) représente donc le projet « mettre Q à 1 » en des termes compréhensibles par le μP et constitue le programme (en langage machine) à charger dans la RAM. Ce programme n'occupe qu'une ligne de la mémoire et est à enregistrer à l'adresse 00 de la mémoire (fig. 1)

Dans tout ce qui va suivre, les entrées WAIT et INTERRUPT sont constamment au niveau 1.

Adresse mémoire	Code de l'instruction	Mnémonique	Commentaire
00	7B	SEQ	1 → Q

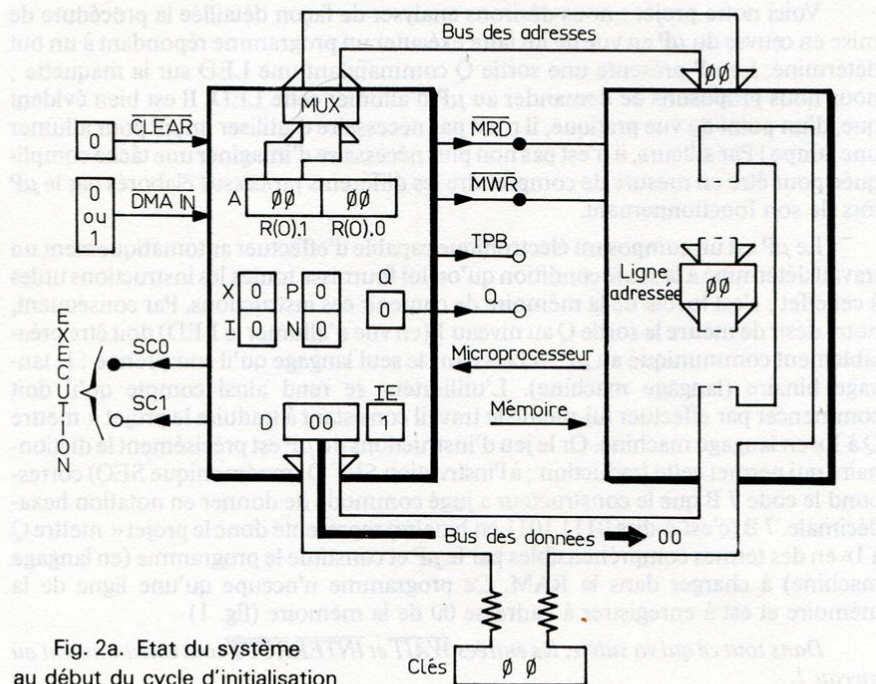
Fig. 1

II. Enregistrement du programme

II - 1. Procédure d'initialisation

Rappelons qu'il va être nécessaire de demander au μP d'établir une voie directe d'accès entre les clés D_7 à D_0 et la RAM pour enregistrer le programme. Or, lors du déroulement d'un cycle DMA au rythme de l'horloge, c'est par construction le registre $R(0)$ qui est pointeur de la mémoire ; comme le contenu de $R(0)$ est aléatoire après la mise sous tension du système, il convient donc avant tout de le mettre à zéro. Ceci est réalisé de la façon suivante :

1. Imposer un niveau 0 sur l'entrée \overline{CLEAR} . Le résultat immédiat de cette

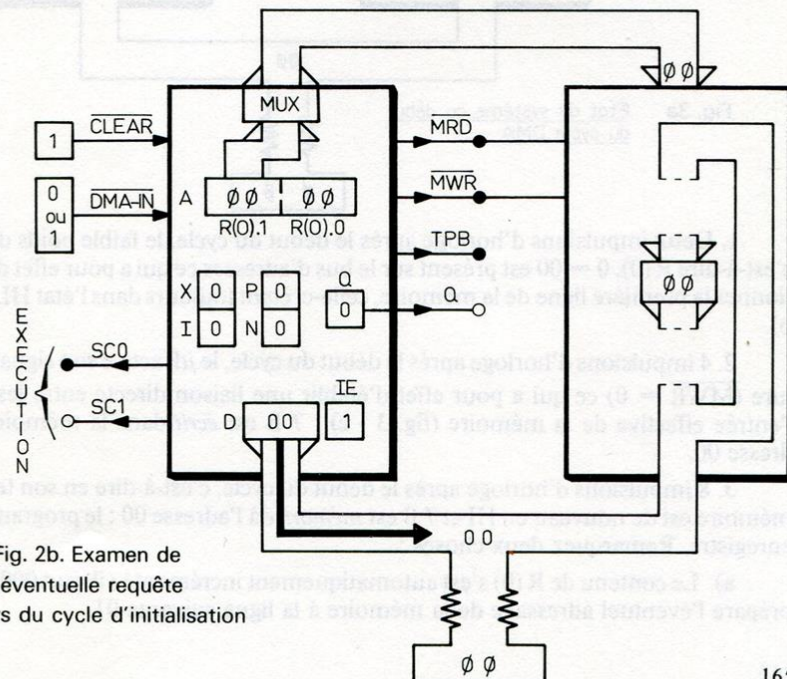


opération est représenté figure 2-a : le μP se met dans l'état « EXÉCUTION » indiquant par ce fait qu'il va entrer dans un cycle d'initialisation dès que $\overline{\text{CLEAR}}$ sera remis au niveau 1 ; par ailleurs, la mémoire est placée dans l'état HI et les registres X, P, I, N, Q internes au μP contiennent zéro. Cependant le contenu du registre d'adresse A est toujours aléatoire. Indiquons que le symbole $\emptyset \emptyset$ représente un octet inconnu, ou bien un octet connu mais dont la valeur nous indiffère.

2. Remettre l'entrée $\overline{\text{CLEAR}}$ au niveau 1 et appliquer les 9 impulsions d'horloge nécessaires à l'exécution du cycle d'initialisation. Au cours de ce cycle, le contenu de R (0) est mis à zéro.

II - 2. Enregistrement du programme

Puisque nous désirons enregistrer un programme, il est nécessaire de placer le μP dans l'état DMA. Ceci est réalisé par une demande préalable (requête) établie en imposant un niveau 0 sur l'entrée $\overline{\text{DMA-IN}}$. Or, rappelons que le μP examine toute requête au cours d'un cycle quelconque lorsque $\text{TPB} = 1$. Par conséquent, si nous désirons que le μP soit dans l'état DMA au terme de la procédure d'initialisation, il convient que l'entrée $\overline{\text{DMA-IN}}$ soit à 0 avant le passage à 1 de TPB au cours du cycle d'initialisation. La figure 2 - b représente le système au cours du cycle d'initialisation



lorsque $TPB = 1$: si $\overline{DMA-IN}$ est alors à 0, ce cycle aboutit à l'entrée du μP dans un cycle DMA annoncé par l'apparition sur les lignes (SC 0, SC 1) de la combinaison (0, 1), et au transfert du contenu égal à zéro du registre R(0) dans le registre A (fig. 3 - a). Le code 7 B (0111 1011 en binaire) que l'utilisateur doit présenter sur les clés est prêt maintenant à être écrit dans la mémoire à l'adresse 00 ; son enregistrement a lieu pendant le déroulement du cycle DMA qui requiert (comme tout cycle autre que le cycle d'initialisation) 8 impulsions d'horloge. Le cycle DMA se décompose essentiellement en 3 mouvements.

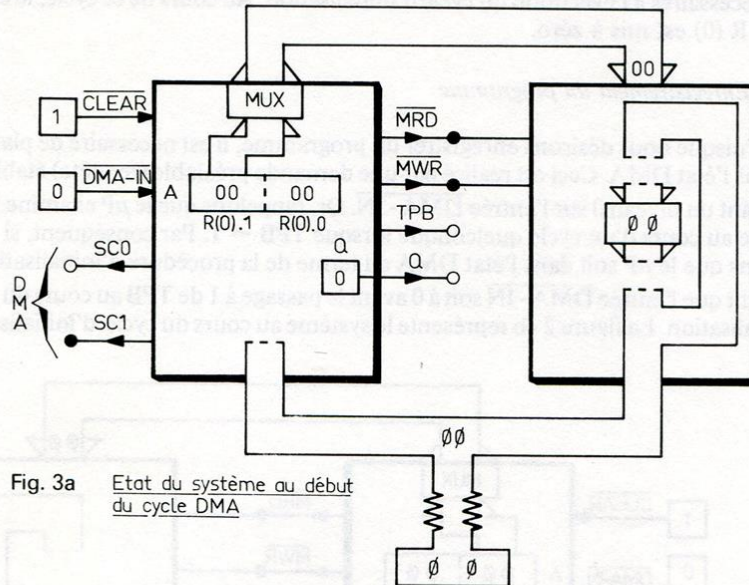


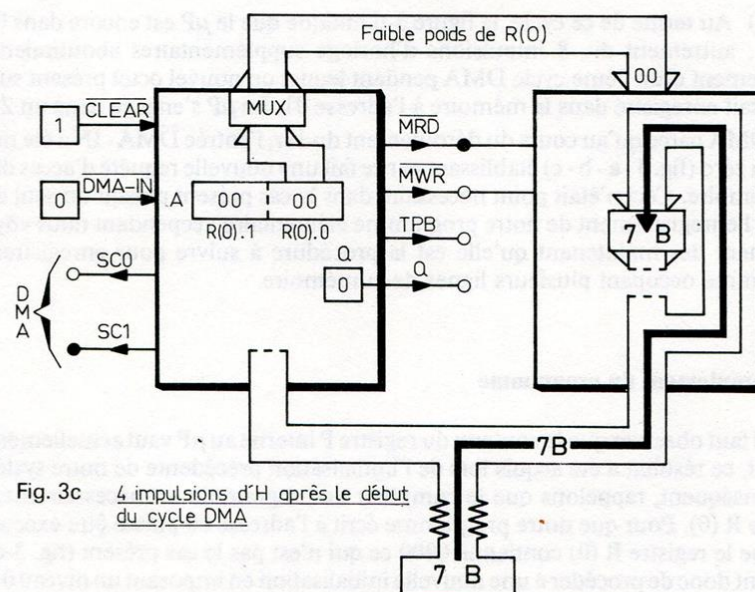
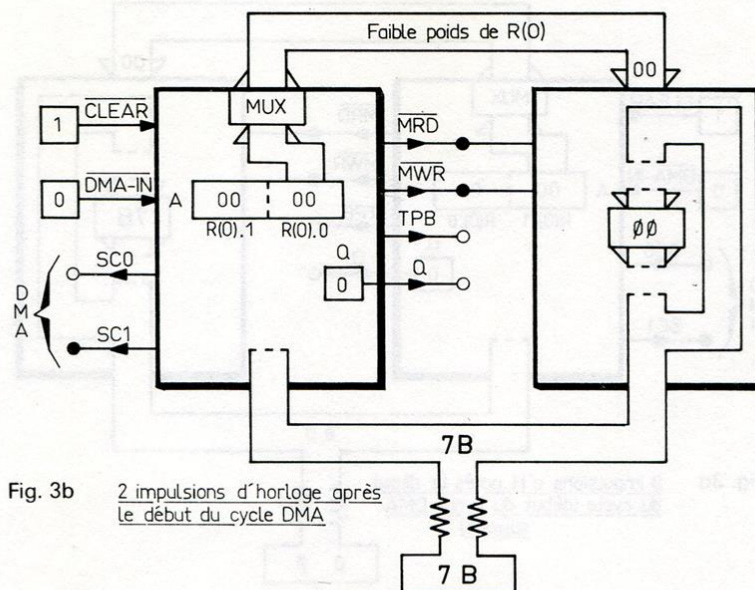
Fig. 3a Etat du système au début du cycle DMA

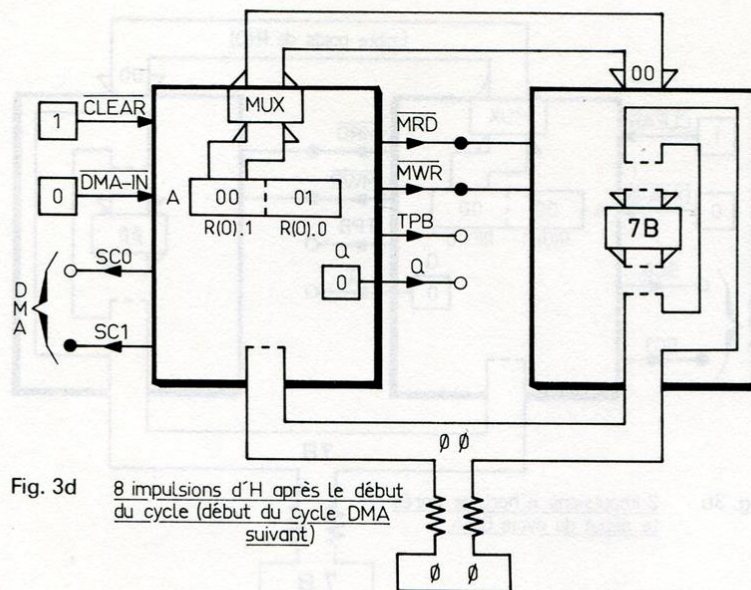
1. Deux impulsions d'horloge après le début du cycle, le faible poids de R(0), c'est-à-dire R(0).0 = 00 est présent sur le bus d'adresses ce qui a pour effet de sélectionner la première ligne de la mémoire, celle-ci étant toujours dans l'état HI. (fig. 3 - b).

2. 4 impulsions d'horloge après le début du cycle, le μP active son signal d'écriture ($\overline{MWR} = 0$) ce qui a pour effet d'établir une liaison directe entre les clés et l'entrée effective de la mémoire (fig. 3 - c) : 7 B est écrit dans la mémoire à l'adresse 00.

3. 8 impulsions d'horloge après le début du cycle, c'est-à-dire en son terme, la mémoire est de nouveau en HI et 7 B est mémorisé à l'adresse 00 : le programme est enregistré. Remarquez deux choses :

a) Le contenu de R(0) s'est automatiquement incrémenté : il vaut 0001 ce qui prépare l'éventuel adressage de la mémoire à la ligne suivante 01.





b) Au terme de ce cycle, la figure 3-d indique que le μP est encore dans l'état DMA ; autrement dit, 8 impulsions d'horloge supplémentaires aboutiraient au déroulement d'un 2ème cycle DMA pendant lequel un nouvel octet présent sur les clés serait enregistré dans la mémoire à l'adresse 01. Le μP s'engage dans un 2ème cycle DMA parce qu'au cours du déroulement du 1er, l'entrée $\overline{\text{DMA-IN}}$ a été maintenue à zéro (fig. 3 - a - b - c) établissant par ce fait une nouvelle requête d'accès direct à la mémoire. Ceci n'était point nécessaire dans le cas présent puisqu'un seul cycle suffit à l'enregistrement de notre programme élémentaire ; cependant nous voyons clairement dès maintenant qu'elle est la procédure à suivre pour enregistrer un programme occupant plusieurs lignes de la mémoire.

III. Déroulement du programme

Il faut observer que le contenu du registre P interne au μP vaut actuellement 0 ; en effet, ce résultat a été acquis lors de l'initialisation précédente de notre système. Par conséquent, rappelons que le compteur de programme est nécessairement le registre R (0). Pour que notre programme écrit à l'adresse 00 puisse être exécuté, il faut que le registre R (0) contienne 0000 ce qui n'est pas le cas présent (fig. 3-d). Il convient donc de procéder à une nouvelle initialisation en imposant un niveau 0 puis

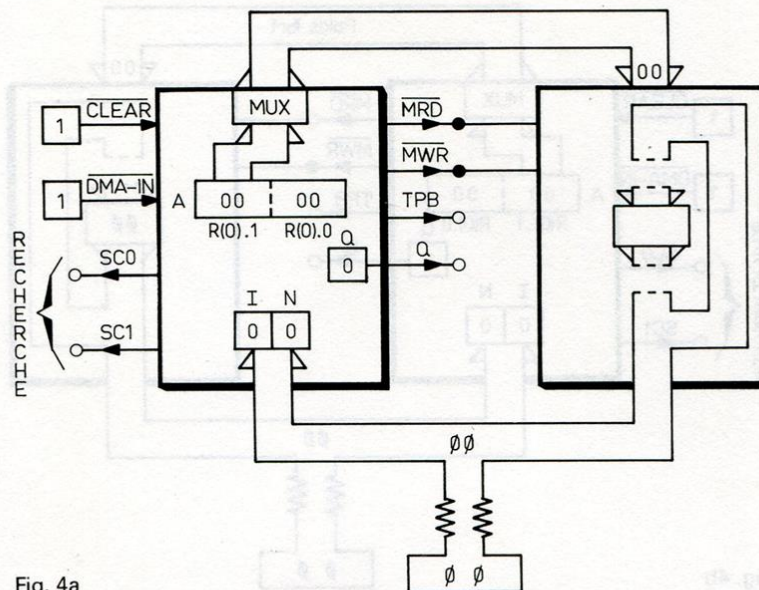


Fig. 4a

un niveau 1 sur l'entrée $\overline{\text{CLEAR}}$ puis en appliquant neuf impulsions d'horloge ; pendant ce cycle d'initialisation, l'entrée $\overline{\text{DMA-IN}}$ doit être au niveau 1 ce qui aboutit, au terme de ce cycle, à l'entrée du μP dans un cycle de recherche (fig. 4-a).

III - 1. Cycle de recherche

Le μP annonce qu'il s'engage dans un cycle de recherche en envoyant sur les lignes (SC0, SC1) la combinaison (0,0). Au début de ce cycle, le contenu du registre A est 0000 et la mémoire est en HI. Par ailleurs, le μP se prépare à faire la lecture du contenu de la mémoire à l'adresse 00 en établissant la liaison entre le bus des données et l'entrée de ses deux registres internes I et N (fig. 4-a).

Le cycle de recherche (qui requiert évidemment 8 impulsions d'horloge) se décompose essentiellement en 4 mouvements.

1. Une impulsion d'horloge après le début de ce cycle, le μP active son signal de lecture ($\overline{\text{MRD}} = 0$) ce qui a pour effet d'établir la liaison entre la sortie effective de la mémoire et le bus des données (fig. 4-b) : la sortie de la mémoire est donc reliée à l'entrée des registres (I, N). Remarquez que c'est le poids fort de R(0) qui est actuellement sur le bus d'adresses et que c'est seulement par hasard que la mémoire est



4. 8 impulsions d'horloge après le début du cycle de recherche, c'est-à-dire à son terme, la mémoire est de nouveau en HI et le μP est prêt à exécuter l'instruction dont il possède le code (fig. 4-e).

III - 2. Cycle d'exécution

Le μP signale qu'il s'engage dans un cycle d'exécution en envoyant sur les lignes (SC 0, SC 1) la combinaison (1, 0). Remarquez que le registre A s'est incrémenté automatiquement et contient donc 0001 au début du cycle d'exécution : le μP

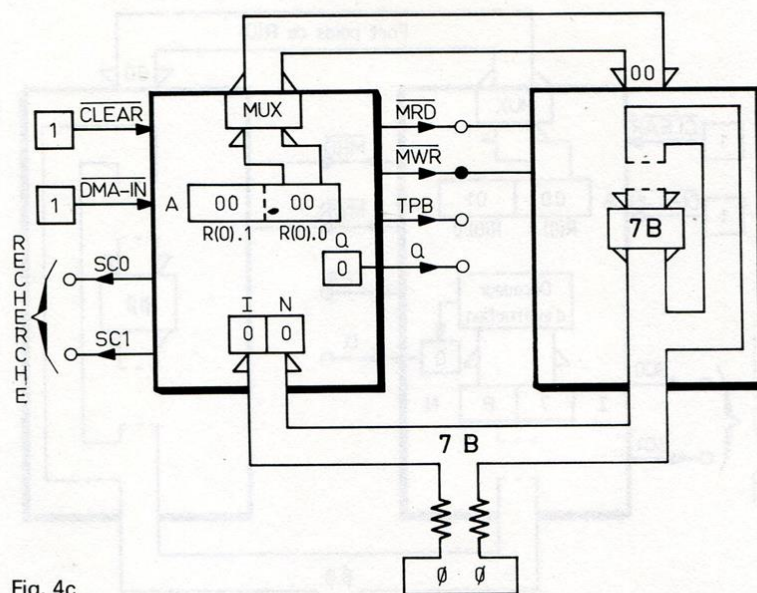


Fig. 4c

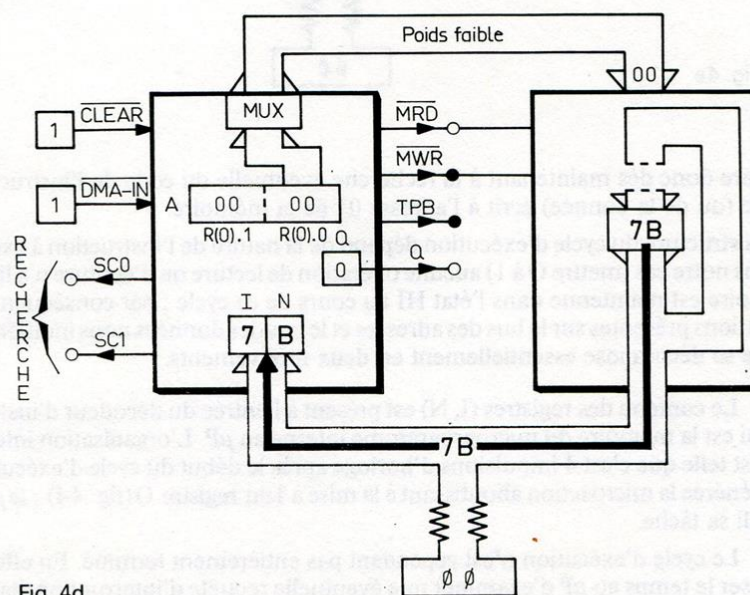
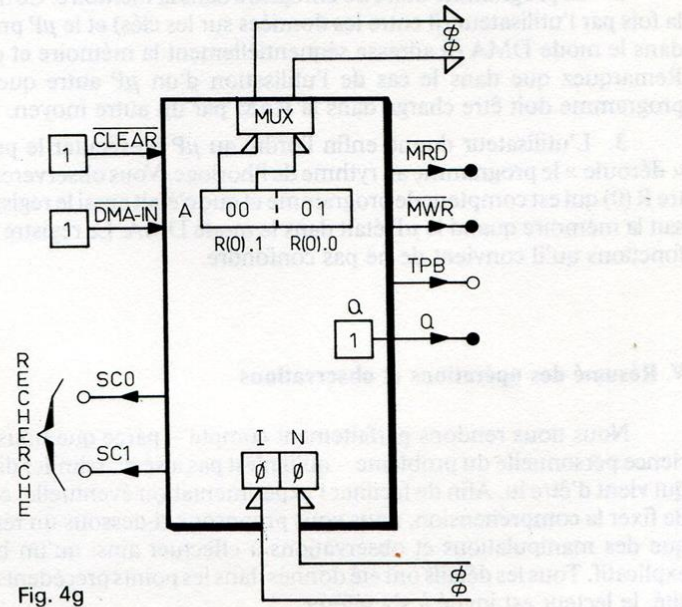
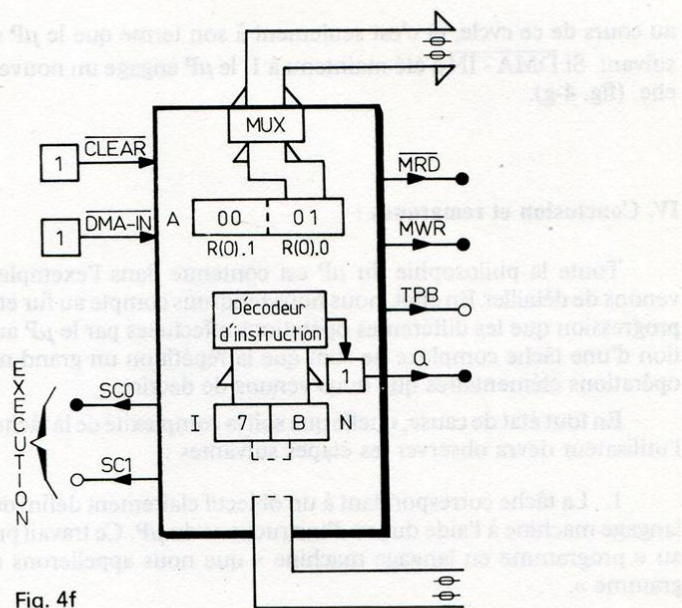


Fig. 4d



La structure du cycle d'exécution dépend de la nature de l'instruction à exécuter. Dans notre cas (mettre Q à 1) aucune opération de lecture ou d'écriture n'a lieu : la mémoire est maintenue dans l'état HI au cours de ce cycle ; par conséquent les informations présentes sur le bus des adresses et le bus des données nous indiffèrent. Ce cycle se décompose essentiellement en deux mouvements.

- 172



au cours de ce cycle, et c'est seulement à son terme que le μP se place dans l'état suivant. Si $\overline{DMA - IN}$ a été maintenu à 1, le μP engage un nouveau cycle de recherche. (fig. 4-g).

IV. Conclusion et remarques :

Toute la philosophie du μP est contenue dans l'exemple simple que nous venons de détailler. En effet, nous nous rendons compte au fur et à mesure de notre progression que les différentes opérations effectuées par le μP au cours de l'exécution d'une tâche complexe ne sont que la répétition un grand nombre de fois des opérations élémentaires que nous venons de décrire.

En tout état de cause, quelle que soit la complexité de la tâche demandée au μP , l'utilisateur devra observer les étapes suivantes :

1. La tâche correspondant à un objectif clairement défini doit être traduite en langage machine à l'aide du jeu d'instructions du μP . Ce travail préliminaire aboutit au « programme en langage machine » que nous appellerons dorénavant « programme ».
2. Le programme doit être enregistré dans la mémoire. Ce travail est effectué à la fois par l'utilisateur (il entre les données sur les clés) et le μP préalablement placé dans le mode DMA (il adresse séquentiellement la mémoire et contrôle son état). Remarquez que dans le cas de l'utilisation d'un μP autre que le COSMAC, le programme doit être chargé dans la RAM par un autre moyen.
3. L'utilisateur donne enfin l'ordre au μP d'exécuter le programme. Le μP « déroule » le programme au rythme de l'horloge. Vous observerez que c'est le registre R(0) qui est compteur de programme et que c'était aussi le registre R(0) qui adressait la mémoire quand le μP était dans le mode DMA. Le registre R(0) a donc deux fonctions qu'il convient de ne pas confondre.

V. Résumé des opérations et observations

Nous nous rendons parfaitement compte – parce que nous avons une expérience personnelle du problème – qu'il n'est pas aisé d'assimiler directement tout ce qui vient d'être lu. Afin de faciliter l'expérimentation éventuelle, et dans tous les cas de fixer la compréhension, nous vous proposons ci-dessous un résumé chronologique des manipulations et observations à effectuer ainsi qu'un bref commentaire explicatif. Tous les détails ont été donnés dans les points précédents : en cas de nécessité, le lecteur est invité à s'y référer.

Ce qu'il faut faire	Ce qu'il faut observer	Commentaires
Mettre l'horloge en pas à pas Faire $\overline{\text{WAIT}} = 1$ et $\overline{\text{INTERRUPT}} = 1$ Faire $\overline{\text{CLEAR}} = 0$, $\overline{\text{DMA}} = 0$ ou 1	$(\text{SC}_1, \text{SC}_0) = (0,1)$ $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,1)$	μP prêt à exécuter un <u>cycle d'initialisation</u> Mémoire en HI De plus $X = P = N = I = Q = 0$ $\text{IE} = 1$
Faire $\overline{\text{DMA}} = 0$ et $\overline{\text{CLEAR}} = 1$ Appliquer 9 impulsions d'horloges	$(\text{SC}_1, \text{SC}_0) = (0,1)$ $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,1)$ TPB passe à 1 et revient à 0 En fin de cycle $(\text{SC}_1, \text{SC}_0) = (1,0)$	μP en <u>exécution d'initialisation</u> . Mémoire en HI Examen et prise en considération de la requête de DMA Le prochain cycle sera un <u>cycle DMA</u> De plus $R(0) = 0$
Présenter le code 7B (01111011) sur les clés D_7 à D_0 Appliquer 8 impulsions d'horloge	$(\text{SC}_1, \text{SC}_0) = (1,0)$ Au début $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,1)$ puis $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,0)$ puis $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,1)$ TPB passe à 1 et revient à 0 En fin de cycle $(\text{SC}_1, \text{SC}_0) = (1,0)$	<u>Cycle DMA</u> Mémoire en HI Mémoire en écriture : 7B s'inscrit à l'adresse 00 Mémoire à nouveau en HI Examen et prise en considération de la requête de DMA Le prochain cycle, s'il a lieu, sera un <u>cycle DMA</u> , $R(0) = 1$
Faire $\overline{\text{CLEAR}} = 0$, $\overline{\text{DMA}} = 0$ ou 1	$(\text{SC}_1, \text{SC}_0) = (0,1)$ $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,1)$	μP prêt à exécuter un <u>cycle d'initialisation</u> Mémoire en HI De plus $X = P = N = I = Q = 0$ $\text{IE} = 1$
Faire $\overline{\text{DMA}} = 1$ et $\overline{\text{CLEAR}} = 1$ Appliquer 9 impulsions d'horloge	$(\text{SC}_1, \text{SC}_0) = (0,1)$ $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,1)$ TPB passe à 1 et revient à 0 En fin de cycle $(\text{SC}_1, \text{SC}_0) = (0,0)$	μP en exécution d'initialisation. Mémoire en HI Pas de prise en considération de requête de DMA ($\overline{\text{DMA}} = 1$) Le prochain cycle sera un <u>cycle de recherche</u> . De plus $R(0) = 0$
Appliquer 8 impulsions d'horloge	$(\text{SC}_1, \text{SC}_0) = (0,0)$ Au début $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,1)$ puis $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (0,1)$ TPB passe alors à 1 et revient à 0 En fin de cycle $(\text{SC}_1, \text{SC}_0) = (0,1)$	<u>Cycle de recherche</u> Mémoire en HI Mémoire en lecture : 7B est présent sur le bus Chargement de 7B dans les registres I et N Le prochain cycle sera un <u>cycle d'exécution</u> . $R(0) = 1$
Appliquer 8 impulsions d'horloge	$(\text{SC}_1, \text{SC}_0) = (0,1)$ $(\overline{\text{MRD}}, \overline{\text{MWR}}) = (1,1)$ Q passe à 1 en milieu de cycle	Cycle <u>d'exécution</u> . Mémoire en HI

Chapitre 11

LE BRANCHEMENT INCONDITIONNEL (ÉTUDE EN PAS-A-PAS)

I. Position du problème

Notre projet est maintenant d'utiliser la sortie Q du μP en vue de réaliser un oscillateur astable. C'est-à-dire que nous voulons demander au μP d'exécuter la succession des tâches suivantes :

« Mettre Q à 1 », « Mettre Q à 0 », « Mettre Q à 1 », « Mettre Q à 0 », et ainsi de suite. Cet objectif est clairement défini par la figure 1 qui représente l'organigramme correspondant. La ligne orientée qui joint les deux rectangles est un axe des temps qui indique l'ordre dans lequel les instructions écrites dans ces rectangles doivent être exécutées.

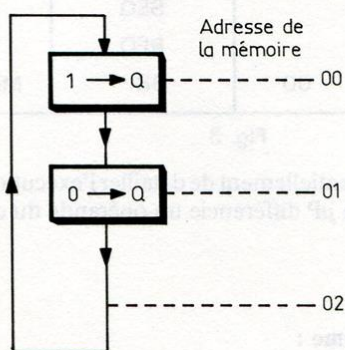


Fig. 1

A chaque instruction correspond une ligne de la mémoire définie par son adresse : le code de l'instruction « mettre Q à 1 » sera écrit à l'adresse 00, le code de l'instruction « mettre Q à 0 » sera écrit à l'adresse 01, et le code de l'instruction permettant le branchement à l'adresse 00 sera écrit à l'adresse 02.

Le jeu d'instructions du μP nous indique que le code du branchement inconditionnel (il doit toujours s'effectuer) est 30. Remarquez qu'il s'agit d'un branchement « court » c'est-à-dire à l'intérieur d'un espace mémoire de 256 mots. Il est bien évident qu'il ne suffit point d'indiquer au μP qu'il doit exécuter un branchement, encore faut-il lui donner l'adresse à laquelle il doit se brancher. Cette adresse (00 en l'occurrence) fait partie intégrante du programme et constitue l'opérande de l'instruction BR ; elle est à écrire à l'adresse 03 de la mémoire.

La figure 2 indique le programme de l'oscillateur astable en langage machine. Pour éviter que n'apparaissent dans la même colonne des codes et des opérands, on a coutume de présenter l'opérande éventuel d'une instruction immédiatement à droite de son code, c'est-à-dire sur la même ligne (fig. 3). Cette présentation facilite la lecture du programme et nous l'adopterons pour la suite.

Adresse	Code ou opérande	Mnémonique	Commentaire
00	7B (code)	SEQ	$1 \rightarrow Q$
01	7A (code)	REQ	$0 \rightarrow Q$
02	30 (code)	BR	$M(R(P)) \rightarrow R(P).0$
03	00 (opérande)		

Fig. 2

Adresse	Code	opérande	Mnémonique	Commentaire
00	7B		SEQ	$1 \rightarrow Q$
01	7A		REQ	$0 \rightarrow Q$
02	30	00	BR	$M(R(P)) \rightarrow R(P).0$

Fig. 3

Nous nous proposons essentiellement de détailler l'exécution du branchement en vue de montrer comment le μP différencie un opérande du code d'une instruction.

II. Enregistrement du programme :

L'enregistrement du programme ne présente pas de difficultés pratiques ni de principe si le lecteur a observé le fait essentiel suivant : au cours de tout cycle DMA, le registre R(0) s'incrémente automatiquement (voir fig. 3 - a - b - c - d du chapitre 10). La procédure à adopter est la suivante : (l'horloge est dans le mode « pas-à-pas »)

1. Faire $\overline{\text{CLEAR}} = 0$ puis $\overline{\text{CLEAR}} = 1$, et appliquer les 9 impulsions d'horloge requises pour le cycle d'initialisation (voir fig. 2 - a et 2 - b du chapitre 10). Il convient d'imposer un niveau 0 sur l'entrée $\overline{\text{DMA-IN}}$ pour qu'en fin de ce cycle, le μP soit placé dans le mode DMA ce qu'il indique en envoyant sur les lignes (SC 0, SC 1) la combinaison (0, 1) (voir fig. 3-a du chapitre 10).

2. Présenter sur les clés le premier mot à écrire, soit 7 B, et appliquer 8 impulsions d'horloge pendant lesquelles on peut vérifier l'écriture de 7 B dans la mémoire à l'adresse 00 (voir fig. 3-c du chapitre 10). L'entrée $\overline{\text{DMA-IN}}$ étant restée au niveau 0, le cycle suivant est encore un cycle DMA ; cependant le contenu de R (0) est maintenant 0001.

3. Présenter sur les clés le mot suivant à écrire, soit 7 A (0111 1010 en binaire) et appliquer 8 autres impulsions d'horloge. On peut vérifier que 4 impulsions d'horloge après le début de ce nouveau cycle DMA, le μP envoie un niveau 0 sur la ligne $\overline{\text{MWR}}$ pendant que le poids faible de R (0) c'est-à-dire 01 est présent sur le bus des adresses : 7 A est écrit dans la mémoire à l'adresse 01. En fin du cycle, R (0) s'est évidemment incrémenté et contient donc 0002.

4. Les deux mots suivants 30 (0011 0000 en binaire) et 00 sont écrits de la même façon respectivement aux adresses 02 et 03 de la mémoire.

III. Déroulement du programme

L'entrée $\overline{\text{DMA-IN}}$ étant au niveau 1, le μP est prêt à exécuter le programme après une nouvelle initialisation ; à l'issue de celle-ci le μP entame un cycle de recherche et le contenu du compteur de programme R (0) est 0000. Le déroulement du programme en « pas-à-pas » permet d'observer les points essentiels suivants :

1. Au cours des 8 impulsions d'horloge requises pour le premier cycle de recherche, le μP mémorise dans ses registres internes (I, N) le code 7 B (voir fig. 4 - a - b - c - d du chapitre 10). En fin de ce cycle, c'est-à-dire au début du cycle d'exécution de l'instruction 7 B, le compteur de programme R (0) s'est incrémenté et contient 0001 (voir fig. 4 - e du chapitre 10). Pendant les 8 impulsions d'horloge supplémentaires qui constituent le cycle d'exécution, nous observons que Q est mis à 1 : la première instruction du programme est exécutée.

2. Le μP entame alors le cycle de recherche de l'instruction suivante. Lors de ce cycle, le μP donne l'ordre à la RAM de se placer en position lecture ($\overline{\text{MRD}} = 0$), le poids faible de R (0), soit 01, étant présent sur le bus des adresses : le mot 7 A est par conséquent mémorisé dans les registres (I, N). En fin de ce cycle de recherche, le compteur de programme s'est incrémenté et son contenu est donc 0002. Pendant le cycle d'exécution de l'instruction 7 A que développe alors le μP , nous observons que Q est mis à 0 : la deuxième instruction du programme est exécutée.



180

Détail du cycle d'exécution de l'instruction de branchement

Depuis le début du chapitre 10 nous n'avons pas jugé utile de faire de distinction entre le registre d'adresses A et le compteur de programme R(0). En effet, pour des raisons de clarté de notre exposé, nous avons considéré que le contenu du registre A était constamment la copie du contenu du registre R(0). Ceci est inexact. En réalité les registres A et R(0) sont des registres latch déclenchés respectivement par les signaux CL_A et TPB (les registres sont transparents si les signaux CL_A et TPB sont au niveau 1). Le μP effectue le branchement de la façon suivante :

1. Au début du cycle d'exécution (fig. 4-a), les registres A et R(0) contiennent 0003 et leurs signaux de commande CL_A et TPB sont au niveau 0. Nous n'avons pas représenté les registres (I, N) qui contiennent actuellement le code 30 ; ce code contient naturellement toutes les informations nécessaires à l'exécution du branchement. En particulier, la liaison entre le bus des données et le demi-registre R(0).0 est établie.

2. La mémoire est placée en position lecture ($\overline{MRD} = 0$) et le faible poids de A, c'est-à-dire 03, est présent sur le bus des adresses. Il en résulte que l'opérande 00 (adresse de branchement) est placé sur le bus des données et est donc présent à l'entrée du demi-registre R(0).0, lequel contient toujours 03. (fig.4-b)

3. Le demi-registre R(0).0 est transparent lorsque TPB = 1 (fig. 4-c) : l'opérande 00 est donc écrit dans R(0).0 tandis que la mémoire est toujours adressée par l'adresse 03.

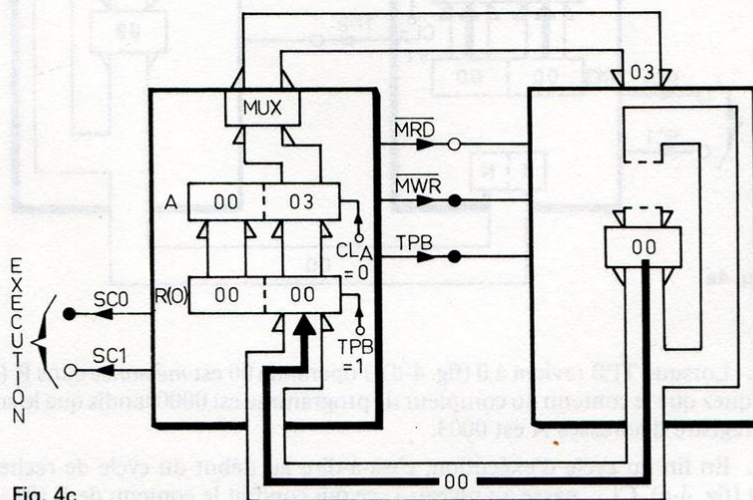
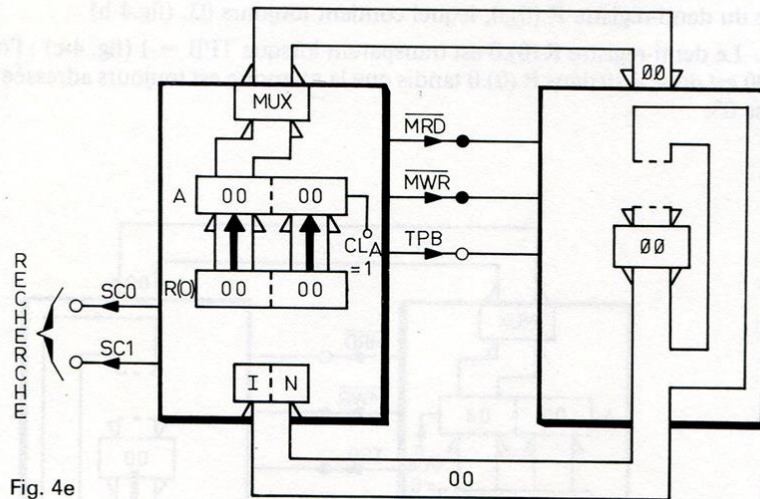
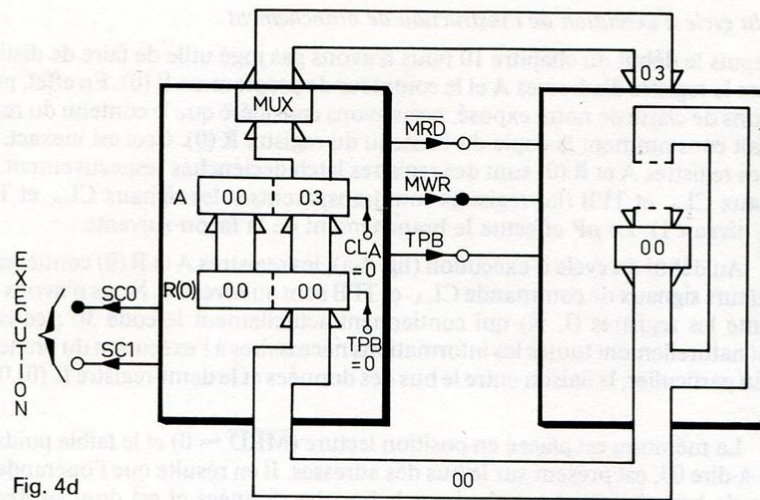


Fig. 4c



4. Lorsque TPB revient à 0 (fig. 4-d), l'opérande 00 est *mémorisé* dans R (0).0. Remarquez que le contenu du compteur de programme est 0000 tandis que le contenu du registre d'adresses A est 0003.

5. En fin du cycle d'exécution, c'est-à-dire au début du cycle de recherche suivant (fig. 4-e), CL_A passe au niveau 1, ce qui conduit le contenu de R (0) à être

$$\frac{1 \text{ MHz}}{48} = \approx 20 \text{ KHz}$$

écrit dans le registre A. Vous remarquerez que seul le poids faible de A est modifié, ce qui est une caractéristique du branchement court. Par ailleurs, la mémoire est mise en HI et l'entrée du demi-registre R (0).0 est « débranchée » du bus des données. Ce cycle de recherche est donc de nouveau le cycle de recherche de l'instruction 7 B écrite à l'adresse 00 de la mémoire. Puis tout recommence au rythme de l'horloge.

Si l'utilisateur place l'horloge dans le mode astable, il constate que la sortie Q passe périodiquement de 0 à 1 ; c'est tous les 6 cycles machine, c'est-à-dire toutes les 6×8 périodes T de l'horloge que Q passe de 0 à 1. Par conséquent, la période T_Q de l'oscillateur ainsi programmé est : $T_Q = 48 T$. *

Le possesseur de la maquette A peut régler T à quelques dixièmes de seconde et observer le déroulement ininterrompu du programme tout en contrôlant l'état du μP (cycle de recherche ou d'exécution), l'adressage de la mémoire (observation du branchement) et l'état logique des signaux utiles (\overline{MRD} , TPB et évidemment Q).

Remarques :

1. L'opérande associé au code de l'instruction BR est exactement l'adresse du branchement. Autrement dit, pour procéder effectivement à ce branchement, le μP doit adresser la mémoire en plaçant sur le bus des adresses un octet exactement égal à l'opérande : un tel adressage est *direct*

2. Dans le jeu d'instructions du μP , le constructeur indique symboliquement de la façon suivante l'opération effectuée par le μP au cours de l'exécution du branchement :

$$M(R(P)) \longrightarrow R(P).0$$

Dans notre cas, P étant nul, cela donne :

$$M(R(0)) \longrightarrow R(0).0$$

En observant que lors du cycle d'exécution du branchement, R (0) contient non pas l'adresse du code du branchement *mais de son opérande*, cette écriture symbolique signifie que « le contenu de la ligne mémoire pointée par R (0) est transféré dans le demi-registre R (0).0 ». Autrement dit, l'opérande est transféré dans R (0).0 ce qui constitue bien un branchement court.

3. La durée de l'état haut ($Q = 1$) de cette bascule astable est de 2 cycles soit 16 périodes d'horloge. La durée de l'état bas ($Q = 0$) est de 4 cycles soit 32 périodes d'horloge. La période est de 6 cycles, soit 48 périodes d'horloge. Le rapport cyclique défini par le rapport entre la durée de l'état haut et la période de la bascule astable est donc égal à $\frac{1}{3}$.

Comment peut-on obtenir un rapport cyclique différent ? La solution la plus simple consiste à ajouter au programme initial (fig. 3) des instructions qui n'ont d'autre effet que de « consommer » du temps :

– instruction « Pas d'opération » (No operation, mnémonique NOP, code C 4). La réalisation de cette instruction particulière (sans effet) se fait en trois cycles, un cycle de recherche et deux cycles d'exécution, soit 24 périodes d'horloge.

– instruction SET Q alors que Q est déjà à 1 et instruction RESET Q alors que Q est à 0

Exemples :

Le déroulement du programme de la figure 5 conduit à un état haut de 4 cycles, un état bas de 4 cycles, donc une période de 8 cycles et un rapport cyclique de $\frac{1}{2}$.

Pour la bascule astable programmée figure 6, la durée de l'état haut est de 2 cycles, la durée de l'état bas de 13 cycles ($3 \times 3 + 2 \times 2$), donc la période est de 15 cycles et le rapport cyclique de $\frac{2}{15}$.

Adresse	Code	Opérande	Mnémonique	Commentaire
00	7B		SEQ	$1 \rightarrow Q$
01	7B		SEQ	pas d'effet
02	7A		REQ	$0 \rightarrow Q$
03	30	00	BR	$M(R(O)) \rightarrow R(O).O$

Fig. 5

Adresse	Code	Opérande	Mnémonique	Commentaire
00	7B		SEQ	$1 \rightarrow Q$
01	7A		REQ	$0 \rightarrow Q$
02	C4		NOP	pas d'opération
03	C4		NOP	pas d'opération
04	C4		NOP	pas d'opération
05	30	00	BR	$M(R(O)) \rightarrow R(O).O$

Fig. 6

Avertissement

Les deux chapitres précédents ont permis, à travers des exemples très simples, de mettre en évidence les processus conduisant d'une part à l'enregistrement, d'autre part au déroulement de programmes élémentaires. Par ailleurs, le lecteur qui a construit la maquette d'étude a pu se familiariser avec son maniement.

Cette étape nous paraît essentielle

Les chapitres qui suivent (chapitre 12 à 16) ont pour objet de développer les techniques qui permettent d'utiliser au mieux les instructions que le microprocesseur COSMAC est capable d'exécuter. Nous nous attacherons à les introduire progressivement, en partant d'exemples simples, concrets destinés à mettre en évidence les problèmes posés et leur résolution.

Chapitre 12

LE BRANCHEMENT CONDITIONNEL

I. Position d'un problème

Le chapitre 11 nous a conduit à l'élaboration d'une bascule astable et nous avons vu comment il est possible d'augmenter la durée de l'état haut ou de l'état bas en introduisant dans le programme des instructions qui sont sans effet (NOP par exemple) mais qui permettent de « consommer du temps ».

Supposons que l'horloge du microprocesseur fournisse un signal de fréquence 1 MHz (rappelons que ceci peut être facilement réalisé en utilisant l'horloge interne au microprocesseur COSMAC et en branchant un cristal 1 MHz entre les broches CL et XTAL). Pour la bascule dont le programme est inscrit sur la figure 3 du chapitre précédent, la durée de l'état haut est de $16\ \mu\text{s}$, celle de l'état bas $32\ \mu\text{s}$. La période est de $48\ \mu\text{s}$. Si l'on se souvient que la réalisation de l'instruction NOP (recherche et exécution) requiert 3 cycles, soit 24 périodes d'horloge ou ici $24\ \mu\text{s}$, on s'aperçoit que, pour obtenir par exemple une durée de l'état haut de $1\ \text{ms} = 1000\ \mu\text{s}$ on est conduit à programmer 41 instructions NOP à la suite de SEP ($\frac{1000 - 16}{24} = 41$). Si l'on veut une durée de l'état bas de 2 ms, il faut programmer 82 fois NOP à la suite de REQ ($\frac{2000 - 32}{24} = 82$). Le programme à enregistrer dans la mémoire est alors celui de la figure 1.

Ce programme utilise 127 lignes de mémoire (à une unité près seulement, la capacité de la mémoire de la maquette A) ! L'enregistrement de ce programme dans la mémoire, s'il n'offre pas plus de difficulté de principe que celui que nous avons effectué au chapitre 11, est une opération longue et fastidieuse.

La *temporisation* réalisée ici grâce à l'addition d'instructions NOP peut être effectuée par un procédé nécessitant l'utilisation d'un espace plus réduit de la mémoire, grâce à l'intervention d'une instruction de *branchement conditionnel*.

Adresse (en décimal)	Code	Opérande	Mnémonique	Commentaire
00	7B		SEQ	$1 \rightarrow Q$
01	C4		NOP	pas d'opération 41 fois
02	C4		NOP	
41	C4		NOP	$0 \rightarrow Q$
42	7A		REQ	
43	C4		NOP	
44	C4		NOP	
				pas d'opération 82 fois
124	C4		NOP	$M(R(0)) \rightarrow R(0).0$
125	30	00	BR	

Fig. 1

II. Le branchement conditionnel (IF)

Comme son nom l'indique, une telle instruction conduit à l'exécution d'un branchement si une condition est remplie. Cette condition peut être (voir liste des instructions) l'état 0 ou 1 du registre DF, de la bascule Q ou d'un drapeau, l'état 0 du contenu du registre D ou l'état « différent de 0 » du contenu du registre D.

Considérons par exemple l'instruction SHORT BRANCH IF D = 0 (branchement court si le contenu du registre D est nul) de mnémonique BZ et le code 32 (en hexadécimal). Supposons que le code 32 de BZ soit enregistré à l'adresse α de la mémoire (fig. 2). Pendant le cycle de *recherche* de cette instruction, 32 est chargé dans les deux registres I et N et le compteur de programme R (P) est incrémenté. Il contient alors $\alpha + 1$.

Pendant le cycle d'*exécution*

– si $D = 0$, le contenu β de la ligne d'adresse $\alpha + 1$ est chargé dans le registre R(P) de manière que, dans le cycle de recherche suivant la ligne d'adresse β soit pointée. C'est exactement ce qui se passe dans le branchement inconditionnel : le lecteur est invité à s'y reporter.

– si $D \neq 0$, le compteur de programme R (P) est incrémenté, tout comme dans un cycle de recherche et son contenu passe donc à $\alpha + 2$. Dans le cycle de recherche

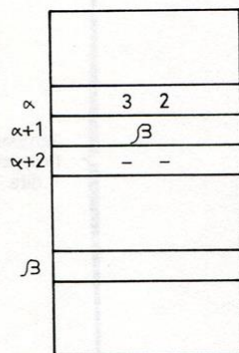


Fig. 2

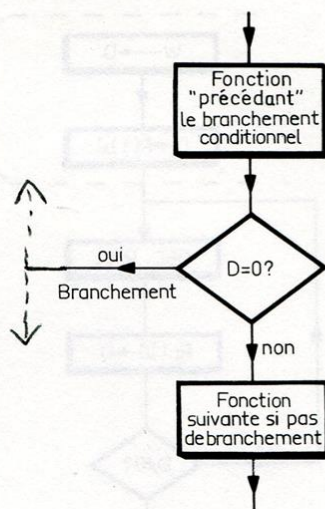


Fig. 3

suivant, c'est donc la ligne d'adresse $\alpha + 2$ qui sera pointée. En d'autres termes, si $D \neq 0$, l'instruction de branchement est ignorée et le programme se continue par l'opération dont le code est enregistré à l'adresse $\alpha + 2$.

Dans l'organigramme d'un programme, le branchement conditionnel faisant intervenir le test $D = 0$ sera représenté comme l'indique la figure 3. Dans le losange est portée la grandeur testée. Le programme s'oriente différemment suivant le résultat du test. Si le test est positif ($D = 0$) le branchement s'effectue, sinon ($D \neq 0$) le programme continue normalement.

III. Utilisation du branchement conditionnel : le temporisateur

III - 1. Principe

Le principe d'une temporisation peut consister en des décréments successives d'un registre chargé initialement par un nombre W . La temporisation est terminée quand le contenu du registre est nul. Comme chaque décrémentation nécessite le même temps, la temporisation est d'autant plus longue que le nombre W est plus grand.

III - 2. Réalisation à l'aide du microprocesseur COSMAC

a) Le registre à décrémenter peut être n'importe lequel des 16 registres R (sauf naturellement celui qui est utilisé comme compteur de programme, c'est-à-dire

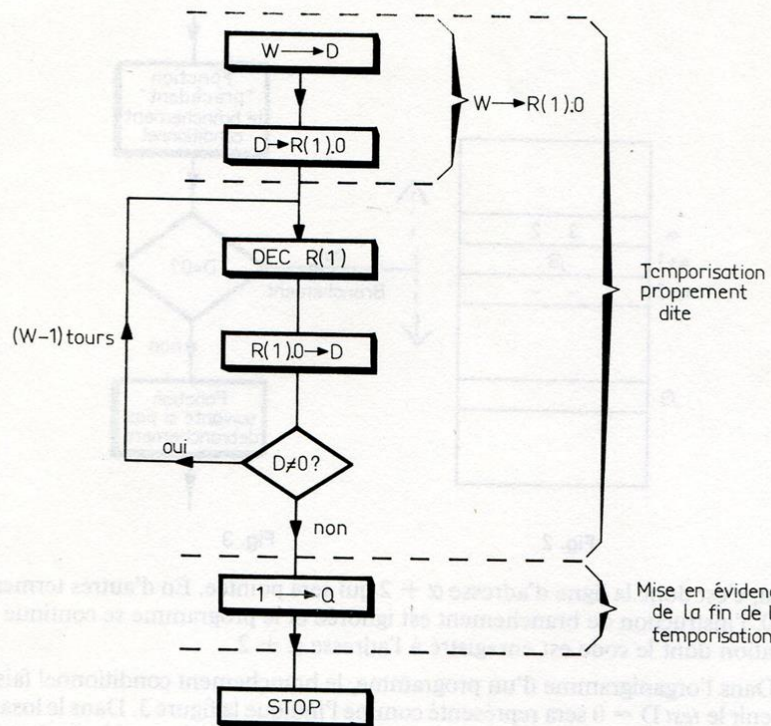


Fig. 4

Adresse	Code	Opérande	Mnémonique	Commentaire
00	F8	W(en hexa)	LDI	$M(R(0)) = W \rightarrow D, R(0) + 1$
02	A1		PLO R(1)	$D = W \rightarrow R(1).0$
03	21		DEC R(1)	$R(1) - 1$
04	81		GLO R(1)	$R(1).0 \rightarrow D$
05	3A	03	BNZ	$M(R(0)) \rightarrow R(0)$ si $D \neq 0$ $R(0) + 1$ si $D = 0$
07	7B		SEQ	$1 \rightarrow Q$
08	00		IDL	mise en attente

Fig. 5

actuellement R (0)). Choisissons par exemple R (1). Pour des temporisations pas trop importantes par rapport à la durée d'un cycle, la décrémentation d'un nombre binaire d'au plus 8 bits peut suffire. Aussi nous pouvons n'utiliser que la partie de faible poids de R (1), soit R (1).0

b) Après décrémentation, il faut faire un branchement conditionnel qui conduit à répéter cette décrémentation uniquement si le contenu du registre R (1).0 est différent de 0. Or l'état (zéro ou pas zéro) d'un registre R n'est pas testable. Le microprocesseur peut par contre effectuer le branchement si $D \neq 0$ (traduire si le contenu de D est différent de 0). Il est nécessaire, après décrémentation du registre R (1), de transférer le contenu du demi-registre R (1).0 dans le registre D. Le branchement est ensuite effectué, grâce à l'instruction BNZ (branchement si $D \neq 0$) si $D = R(1).0 \neq 0$.

c) Il importe évidemment de charger initialement le demi-registre R(1).0 par le nombre W à décrémenter, caractérisant la durée de la temporisation. Cette opération ne peut pas se faire directement. Le nombre Winscrit en mémoire est d'abord chargé dans le registre D (instruction LOAD IMMEDIATE, LDI) dont le contenu (W donc) est transféré dans la partie de faible poids du registre R (1) (instruction PUT LOW REGISTER 1, PLO). On aboutit à l'organigramme de la figure 4.

d) Dans le but de permettre au possesseur de la maquette d'expérimenter la temporisation, nous avons fait suivre cette temporisation par une mise à 1 de la bascule Q destinée à visualiser la fin de la temporisation.

Le blocage du compteur de programme est réalisé grâce à l'instruction IDLE qui place le microprocesseur en état d'attente. Nous reviendrons plus loin sur cette instruction.

III - 3. Ecriture du programme

Dans le but de vérifier aisément expérimentalement le programme temporisateur, écrivons ce programme à partir de l'adresse 00 de la mémoire (fig. 5).

Ce programme appelle les remarques suivantes :

- Les adresses 01 et 06 ne sont pas explicitement écrites. Les lignes correspondantes de la RAM supportent néanmoins les nombres W et 03 (écrits dans le programme en hexadécimal) qui sont les *opérandes* respectifs des instructions LDI et BNZ.

- Ces opérandes ne sont pas de même nature : l'opérande W de LDI est une *donnée* ; l'opérande 03 de BNZ est une *adresse*.

III - 4. Calcul de la durée de temporisation (se reporter à la figure 4)

Décidons pour ce calcul que W représente le nombre, maintenant *en décimal*, que l'on charge initialement dans le demi-registre R (1).0.

Remarquons que les instructions décrémentation du registre R(1), transfert de R(1).0 dans le registre D, branchement conditionnel, sont utilisées W fois. R(1).0 qui contient initialement ce nombre est appelé *compteur de boucles* (on notera cependant que la boucle complète n'est décrite que W - 1 fois : le branchement ne s'effectue que W - 1 fois). La réalisation de ces trois instructions nécessite $3 \times 2 = 6$ cycles (trois cycles de recherche et trois cycles d'exécution). La réalisation du chargement de W dans le registre R(1).0 requiert elle $2 \times 2 = 4$ cycles.

La durée de temporisation est donc $\theta = (6.W + 4)$ cycles = $(6.W + 4) 8 T$ où T désigne la période de l'horloge. Si $W \gg 1$, $\theta \approx 48 W.T$

III - 5. Enregistrement et déroulement du programme

a) Le possesseur de la maquette A (et éventuellement de la maquette B) est invité à enregistrer ce programme dans la RAM en choisissant, par exemple, W(décimal) = 15, soit W(hexadécimal) = 0 F ou W(binaire) = 0000 1111. Il doit pour cela fixer \overline{DMA} à 0, laisser \overline{WAIT} et $\overline{INTERRUPT}$ à 1 et faire $\overline{CLEAR} = 0$, puis $\overline{CLEAR} = 1$, appuyer 9 fois sur le bouton poussoir de fonctionnement pas-à-pas de l'horloge (initialisation), présenter F 8 (en binaire) sur les clés de programmation, appuyer 8 fois sur le bouton poussoir d'horloge, présenter W(binaire) sur les clés de programmation, appuyer 8 fois sur le bouton poussoir d'horloge et ainsi de suite jusqu'à la présentation du code 00 de IDL et son enregistrement à l'adresse 08. Pour plus de détails, revoir les chapitres précédents.

Si la maquette B est installée, les adresses 00 à 08 apparaissent successivement sur les afficheurs d'adresses.

b) Faire dérouler le programme, soit en utilisant le mode pas-à-pas, soit en utilisant le mode astable mais en prenant bien soin dans ce cas de choisir une fréquence d'horloge suffisamment faible pour que l'on puisse suivre le déroulement des opérations. Pour cela, faire $\overline{CLEAR} = 0$, $\overline{DMA} = 1$, mettre l'horloge en mode astable, revenir à $\overline{CLEAR} = 1$: le programme se déroule.

On peut suivre les adresses successives de la mémoire et en particulier constater les sauts de l'adresse 06 à l'adresse 03, en hexadécimal sur la maquette B, en binaire sur la maquette A. On peut aussi suivre l'évolution du contenu du compteur de boucle R(1).0. En effet, lors de l'exécution de la décrémentation du registre R(1), le contenu du registre R(1) est d'abord chargé sur le registre A et son octet de poids fort puis son octet de poids faible apparaît sur le bus des adresses et peut donc être visualisé.

Quand le décomptage de R(1) est terminé, Q passe à 1 et la LED correspondante s'allume.

Il est intéressant de vérifier la durée de temporisation : mesurer la durée entre la mise à 1 de \overline{CLEAR} et le passage de 1 de Q. Comparer au calcul en tenant compte

(si la mesure est précise) de la durée du cycle d'initialisation (9 périodes d'horloge) et du temps nécessaire à la mise à 1 du registre Q ($8 + 4$ périodes)

IV. Bascule astable dont la durée des états haut et bas est grande devant un cycle

Le problème a été posé dès le début de ce chapitre. Sa solution consiste en la programmation de deux temporisations destinées à maintenir respectivement la bascule Q dans l'état haut ou dans l'état bas suivant l'organigramme de la figure 6.

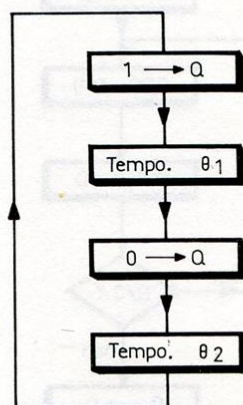


Fig. 6

Les durées des temporisations θ_1 et θ_2 sont fonction des nombres W_1 et W_2 que l'on aura chargés initialement dans les compteurs de boucle qui peuvent être le même registre R (1).0 ou deux registres différents R (1).0 et R (2).0 (notre exemple). L'organigramme complet est celui de la figure 7. La figure 8 donne le programme qui lui correspond en langage machine (les adresses sont en hexadécimal). Ce programme utilise 17 lignes de mémoire : comparez avec le programme de la figure 1.

Le possesseur de la maquette A est invité à enregistrer ce programme puis à le faire se dérouler (horloge en mode astable) à une vitesse suffisamment faible pour suivre les différentes étapes, comme pour le temporisateur du point III. Il peut vérifier que les durées des états haut et bas de la bascule astable qu'il a synthétisée correspondent aux durées qu'il est en mesure de calculer à partir des nombres W_1 et W_2 et de la période d'horloge qu'il a choisis.

Application numérique

On veut réaliser : – durée de l'état haut : 1 ms
– durée de l'état bas : 2 ms

L'horloge ayant une fréquence de 1 MHz (période $1 \mu s$) fixée par un quartz.

La durée d'un cycle est de $8 \mu s$. La temporisation θ_1 doit durer $(1000 - 2 \times 8) \mu s$

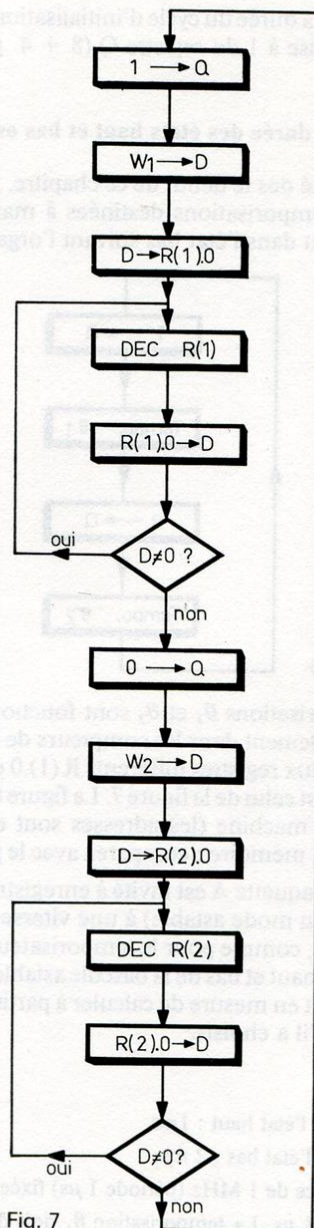


Fig. 7

Adresse	Code	Opérande	Mnémonique	Commentaire
00	7B		SEQ	$1 \rightarrow Q$
01	F8	W_1 (hexa)	LDI	$M(R(0)) = W_1 \rightarrow D$
03	A1		PLO R(1)	$D = W_1 \rightarrow R(1).0$
04	21		DEC R(1)	$R(1) - 1$
05	81		GLO R(1)	$R(1).0 \rightarrow D$
06	3A	04	BNZ	$M(R(0)) = 04 \rightarrow R(0)$ si $D \neq 0$; $R(0) + 1$ si $D = 0$
08	7A		REQ	$O \rightarrow Q$
09	F8	W_2 (hexa)	LDI	$M(R(0)) = W_2 \rightarrow D$
0B	A2		PLO R(2)	$D = W_2 \rightarrow R(2).0$
0C	22		DEC R(2)	$R(2) - 1$
0D	82		GLO R(2)	$R(2).0 \rightarrow D$
0E	3A	0C	BNZ	$M(R(0)) = 0C \rightarrow R(0)$ si $D \neq 0$; $R(0) + 1$ si $D = 0$
10	30	00	BR	$M(R(0)) = 00 \rightarrow R(0)$

Fig. 8

$= 984 \mu s$ soit $984 : 8 = 123$ cycles. Il faut donc choisir $W_1 = \frac{123 - 4}{6} \approx 20$ par excès (la durée de l'état haut sera très légèrement supérieur à 1 ms). La temporisation θ_2 doit durer $(2000 - 4 \times 8) \mu s$ soit 246 cycles. Il faut donc choisir $W_2 = \frac{246 - 4}{6} \approx 40$ par défaut (durée de l'état bas légèrement inférieure à 2 ms)

Si l'on dispose d'un quartz 1 MHz et d'un instrument (oscilloscope par exemple) permettant la mesure de ces durées, il est intéressant de procéder à une vérification.

V. Assimilation : exercices de programmation.

Il s'agit ici de consolider les connaissances acquises. Les exercices proposés font appel aux instructions que nous avons déjà rencontrées ou à des instructions voisines (PHI, GHI, B1)

V - 1. Temporisation longue

a) position du problème

Les nombres W_1 et W_2 que nous venons de trouver sont inférieurs à $2^8 = 256$ et peuvent être enregistrés sous forme binaire dans les demi-registres $R(1).0$ et $R(2).0$. On conçoit que des

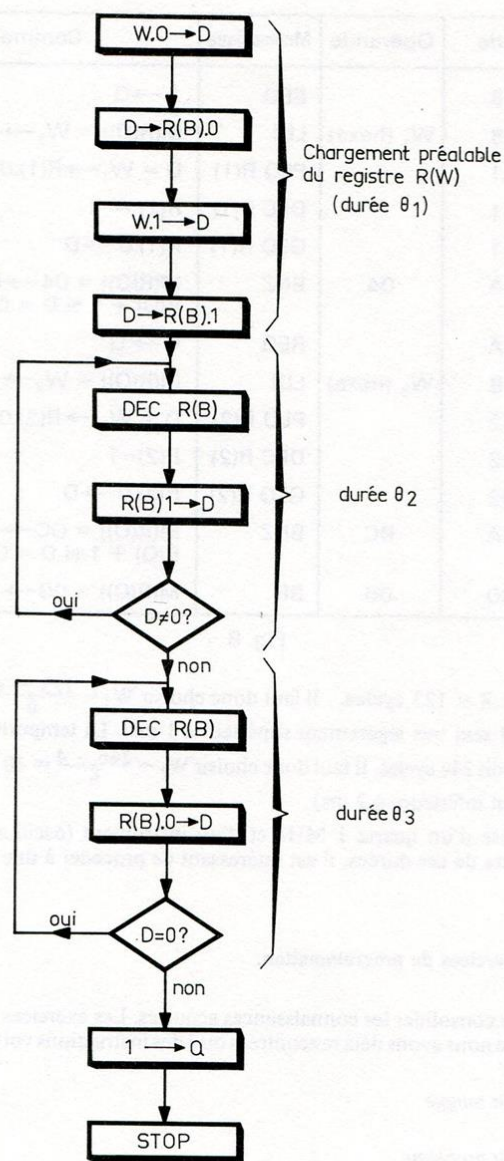


Fig. 10

temporisations beaucoup plus longues nécessiteraient l'enregistrement dans R(1) ou R(2) de nombres pouvant facilement dépasser 256. La capacité de ces registres étant de 16 bits, les nombres peuvent aller jusqu'à $65535 = 2^{16} - 1$.

Nous nous proposons de programmer un temporisateur dont le compteur de boucle, R(B) par exemple, est initialement chargé par W tel que $256 \leq W \text{ (décimal)} \leq 65535$. La partie R(B).0 de R(B) contient initialement W.0 (binaire) correspondant aux 8 bits de plus faible poids de W (binaire). La partie R(B).1 contient initialement W.1 (binaire) correspondant aux bits de plus fort poids de W (binaire). (voir fig. 9).

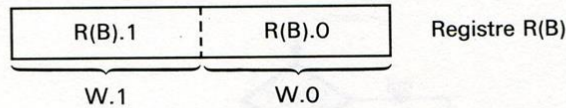


Fig. 9

En ce qui concerne l'écriture du programme, une petite difficulté vient du fait que, si la décrémentation s'opère bien sur l'ensemble du registre, il n'est pas possible de tester directement le contenu (0 ou pas) du registre R(B) de 16 bits puisque le test se fait par l'intermédiaire du registre D qui n'a que 8 bits.

b) solution proposée

Elle est traduite par l'organigramme de la figure 10. Elle consiste à :

- charger le registre R(B) par W
- décrémentation le registre R(B) et recommencer tant que R(B).1 n'est pas à 0

Adresse	Code	Opérande	Mnémonique	Commentaire
00	F8	W.O(hexa)	LDI	$M(R(0)) = W.O \rightarrow D$
02	AB		PLO R(B)	$D = W.O \rightarrow R(B).O$
03	F8	W.1(hexa)	LDI	$M(R(0)) = W.1 \rightarrow D$
05	BB		PHI R(B)	$D \rightarrow R(B).1$
06	2B		DEC R(B)	$R(B) - 1$
07	9B		GHI R(B)	$R(B).1 \rightarrow D$
08	3A	06	BNZ	$M(R(0)) = 06 \rightarrow R(0) \text{ si } D \neq 0; R(0) + 1 \text{ si } D = 0$
0A	2B		DEC R(B)	$R(B) - 1$
0B	8B		GLO R(B)	$R(B).O \rightarrow D$
0C	3A	0A	BNZ	$M(R(0)) = 3A \rightarrow R(0) \text{ si } D \neq 0; R(0) + 1 \text{ si } D = 0$
0E	7B		SEQ	$1 \rightarrow Q$
0F	00		IDL	Attente

Fig. 11

- décrémenter le registre R (B) et recommencer tant que R (B).0 n'est pas à 0
- quand R (B).0 = 0, mettre la bascule Q à l'état haut de manière à pouvoir visualiser la fin de la temporisation et arrêter le déroulement

La figure 11 indique le programme à inscrire en mémoire.

Nous laissons au lecteur le soin d'essayer le programme proposé (s'il possède la maquette), de montrer qu'avec 2 registres de 16 bits on peut réaliser des temporisations encore plus longues (établir l'organigramme).

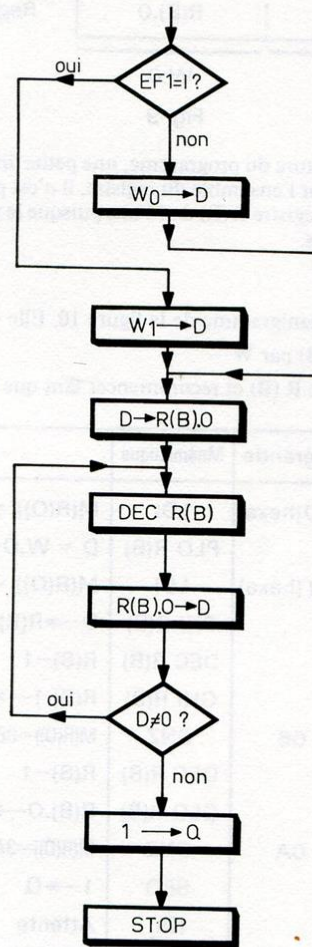


Fig. 12

Remarque :

L'exécution du programme temporisateur dont l'organigramme est représenté sur la figure 10 nécessite :

$\theta_1 = 8$ cycles pour le chargement préalable du registre R (B)

$\theta_2 = [(W_0 + 1) + 256 (W_1 - 1)]$. 6 cycles pour franchir le cap du premier test

$\theta_3 = 255$. 6 cycles supplémentaires pour franchir le cap du deuxième test

La durée de temporisation est donc :

$$\theta = \theta_1 + \theta_2 + \theta_3 = [(W_0 + 256 W_1) \cdot 6 + 8] \text{ cycles}$$

Il est à noter que la séquence de durée θ_3 n'apporte rien d'intéressant et qu'elle peut donc être supprimée. Alors $\theta = \theta_1 + \theta_2 \approx \theta_2$. Si l'on désire une expression simple de la durée de temporisation, on peut choisir $W_0 = 255$ (soit FF en hexadécimal) et alors :

$$\theta \approx \theta_2 = 256 \cdot 6 \cdot W_1 \text{ cycles} = 256 \cdot 48 \cdot W_1 T$$

V - 2. Temporisation de durée conditionnelle

a) Position du problème

On désire obtenir une temporisation de durée θ_1 ou bien de durée θ_0 suivant l'état logique 1 ou 0 d'un drapeau (EF 1 par exemple). On choisira R (B) comme compteur de boucles et l'on supposera que les nombres W_1 et W_2 à charger initialement dans R (B) pour

Adresse	Code	Opérande	Mnémonique	Commentaire
00	34	06	B1	$M(R(O)) = 06 \rightarrow R(O)$ si EF1 = 1, $R(O) + 1$ si EF1 = 0
02	F8	W_0 (hexa)	LDI	$M(R(O)) = W_0 \rightarrow D$
04	30	08	BR	$M(R(O)) = 08 \rightarrow R(O)$
06	F8	W_1 (hexa)	LDI	$M(R(O)) = W_1 \rightarrow D$
08	AB		PLO R(B)	$D \rightarrow R(B).O$
09	2B		DEC R(B)	$R(B) - 1$
0A	8B		GLO R(B)	$R(B).O \rightarrow D$
0B	3A	09	BNZ	$M(R(O)) = 09 \rightarrow R(O)$ si $D \neq 0$, $R(O) + 1$ si $D = 0$
0D	7B		SEQ	$1 \rightarrow Q$
0E	00		IDL	attente

Fig. 13

obtenir respectivement les durées θ_1 et θ_0 sont inférieurs à 256 (décimal). Seul le demi-registre R (B).0 est donc utilisé.

b) solution proposée. (il est possible d'en trouver d'autres !)

Il faut s'arranger pour charger W_1 dans le registre D si $EF\ 1 = 1$ et charger W_0 dans le registre D si $EF\ 1 = 0$ puis dans l'un ou l'autre des deux cas transférer le contenu (W_1 ou W_0) du registre D dans le registre R (B).0. Ensuite, on revient au programme temporisateur habituel.

L'organigramme de la figure 12 fixe le principe du programme donné figure 13

Lors de l'essai éventuel de ce programme, il faut être attentif au fait que les entrées de commande du microprocesseur ne sont pas directement $EF\ 1$, $EF\ 2$, $EF\ 3$, $EF\ 4$ mais $\overline{EF}\ 1$, $\overline{EF}\ 2$, $\overline{EF}\ 3$ et $\overline{EF}\ 4$. Le drapeau $EF\ 1$ est donc à l'état logique 1 si l'on porte l'entrée $\overline{EF}\ 1$ au niveau 0 et réciproquement

Note aux utilisateurs de la maquette

Pour enregistrer les différents programmes élaborés jusqu'ici, vous avez peut-être utilisé uniquement la méthode « pas-à-pas » qui a été suggérée dans les deux chapitres précédents et au début du chapitre présent. Cela vous a permis de suivre de très près la chronologie des événements. Vous vous êtes sans doute rendu compte que cette méthode de chargement de la mémoire devient vite fastidieuse. Aussi nous vous encourageons à utiliser dès maintenant la *procédure spéciale de chargement d'un programme* dans la mémoire, décrite entièrement et justifiée dans la 2ème partie de cet ouvrage (paragraphe VIII, 1 du chapitre 7) et qui conduit au mode opératoire suivant :

- 1) \overline{WAIT} étant à 1, mettre \overline{CLEAR} à 0
- 2) Mettre l'horloge dans le mode astable (si ce n'est pas déjà le cas)
- 3) Effectuer la liaison par un cordon entre la sortie \overline{SR} de la bascule JK et la douille de l'entrée de commande $\overline{DMA - IN}$.
- 4) Vérifier que $\overline{SR} = 1$. Si ce n'est pas le cas, faire $\overline{WAIT} = 0$ puis revenir à $\overline{WAIT} = 1$ ce qui a pour conséquence de mettre \overline{SR} à 1.
- 5) Mettre \overline{WAIT} à 0. On est alors en mode LOAD.
- 6) Présenter sur les clés de programmation D_7 à D_0 l'octet à enregistrer à l'adresse 00 de la mémoire.
- 7) Appuyer sur le bouton poussoir : l'octet présenté s'inscrit à l'adresse 00 lisible sur les LED d'affichage de l'adresse (maquette A). L'adresse 00 est lisible sur les afficheurs hexadécimaux d'adresse de la maquette B et le nombre que l'on vient d'inscrire à cette adresse est lisible sur les afficheurs de données de la maquette B.
- 8) Présenter sur les clés D_7 à D_0 l'octet à enregistrer à l'adresse 01.
- 9) Appuyer sur le bouton poussoir.
- 10) Présenter sur les clés D_7 à D_0 l'octet à enregistrer à l'adresse 02.

11) Appuyer sur le bouton poussoir....
.....et ainsi de suite jusqu'à enregistrement complet du programme.

Si, après avoir chargé la mémoire, vous éprouvez le besoin d'effectuer la *vérification de son contenu*, vous devez :

- 1) Mettre l'inverseur RAM – ROM sur la position ROM.
- 2) Faire $\overline{\text{WAIT}} = 1$ puis $\overline{\text{WAIT}} = 0$ (initialisation du pointeur R (0)).
- 3) Appuyer sur le bouton poussoir pour visualiser sur les afficheurs de données, les octets enregistrés aux adresses successives lisibles simultanément sur les afficheurs d'adresses.

S'il faut effectuer *une modification* de l'octet inscrit à l'adresse \mathcal{A} , reprendre les 3 points précédents mais lorsque l'adresse $\mathcal{A} - 1$ est apparue :

- 1) Mettre l'inverseur sur la position RAM.
- 2) Présenter sur les clés l'octet à enregistrer à l'adresse \mathcal{A} .
- 3) Appuyer une fois sur le bouton poussoir.
- 4) Remettre l'inverseur en position ROM, si l'on veut poursuivre la vérification au delà de l'adresse \mathcal{A} .
- 5) Appuyer sur le bouton poussoir pour éventuellement achever la vérification.

Le programme étant maintenant convenablement enregistré, pour provoquer *son déroulement* il faut :

- 1) Oter la liaison de $\overline{\text{SR}}$ à $\overline{\text{DMA-IN}}$ si l'on désire faire dérouler le programme pas à pas.
- 2) Mettre (si ce n'est pas déjà fait) l'inverseur en position RAM.
- 3) Mettre l'horloge dans le mode de fonctionnement souhaité (astable ou pas à pas).
- 4) Mettre $\overline{\text{WAIT}}$ à 1 puis $\overline{\text{CLEAR}}$ à 1. Après initialisation, le programme se déroule au rythme de l'horloge.

Nous avons établi cette énumération dans le seul but de faciliter votre utilisation de la maquette d'étude. Il serait souhaitable de parvenir à une compréhension raisonnée de ce « mode d'emploi ».

Nous vous conseillons de reprendre quelques uns des programmes établis et de les expérimenter en utilisant systématiquement le mode LOAD.

Chapitre 13

LES SOUS-PROGRAMMES

I. Mise en évidence de l'utilité des sous-programmes

I.1. Position d'un problème : réalisation d'une alarme

Nous nous proposons d'utiliser notre système à microprocesseur COSMAC pour réaliser un système d'alarme destiné à aider la surveillance d'une porte, dont le fonctionnement doit être le suivant :

- la mise en service de l'alarme est assurée par la manœuvre d'un commutateur marche-arrêt.
- quand cette mise en service est effectuée, l'opérateur dispose de $\theta_1 = 40s$ pour fermer la porte à surveiller sans que l'alarme ne se déclenche.
- l'ouverture de la porte par un intrus doit déclencher une sonnerie intermittente, succession de sons et de silences de durées respectives $\theta_3 = 1s$ et $\theta_4 = 0,5s$.
- cette sonnerie ne doit commencer que $\theta_2 = 20s$ après l'ouverture de la porte, afin de permettre à l'opérateur qui a lui-même ouvert la porte de mettre l'alarme hors-service en manœuvrant le commutateur marche-arrêt.
- l'arrêt de l'alarme quand elle est déclenchée se fait à tout instant en mettant le commutateur sur la position arrêt.

Dans la mise en œuvre, le commutateur marche-arrêt peut être le commutateur \overline{CLEAR} : mettre \overline{CLEAR} à 0 lorsque $\overline{WAIT} = 1$ laisse le microprocesseur dans un état d'attente ; faire alors $\overline{CLEAR} = 1$ entraîne son initialisation puis le déroulement du programme.

La détection de l'intrus est aisément réalisable en installant sur la porte un contact qui impose $\overline{EF1} = 0$ si la porte est fermée et $\overline{EF1} = 1$ si la porte est ouverte (fig. 1).

La sonnerie intermittente est commandée par une bascule astable réalisée par le microprocesseur suivant le principe étudié dans le chapitre précédent.

Cette alarme fonctionne alors suivant l'organigramme de la figure 2.

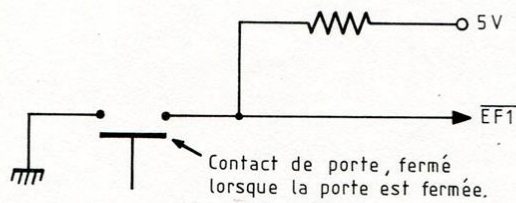


Fig. 1

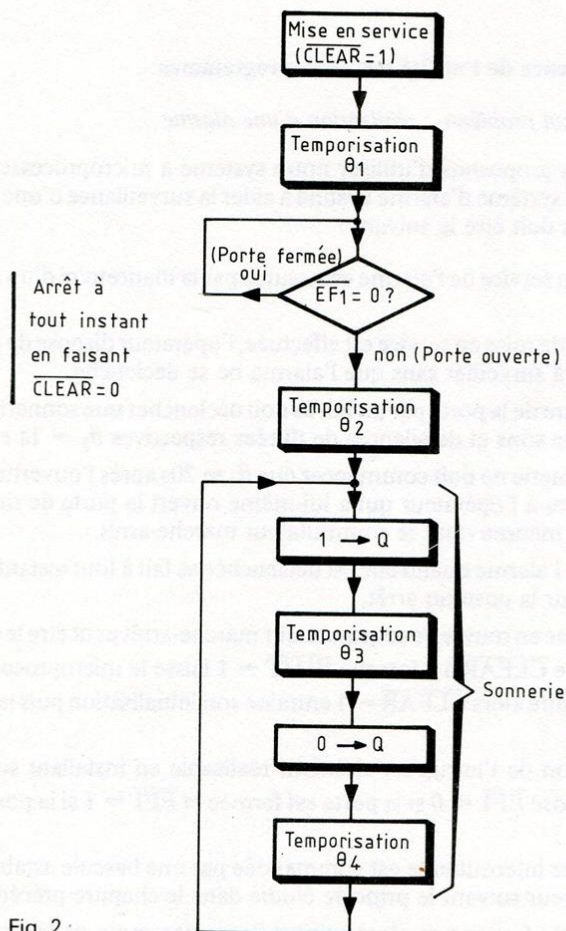


Fig. 2

I. 2. Utilité d'un sous-programme

L'examen de l'organigramme de fonctionnement fait apparaître que l'alarme nécessite quatre temporisations. Pour établir le programme correspondant, une première solution consiste à écrire quatre fois le programme temporisateur. Cette solution, de mise en œuvre simple, a le défaut de consommer beaucoup d'espace mémoire. Ce défaut est accentué lorsque la séquence d'instructions à répéter plusieurs fois est plus longue et le nombre de répétitions est plus élevé.

Une deuxième solution est d'écrire dans la mémoire externe le programme temporisateur *une seule fois* et de faire appel à ce programme (alors nommé *sous-programme*) chaque fois que l'on en a besoin. Quand le sous-programme appelé est exécuté, il est nécessaire de retourner au programme qui l'a appelé (*programme principal*).

II. Mise en œuvre d'un sous-programme

II. 1. Changement de registre compteur de programme

On sait que, après initialisation, le déroulement d'un programme commence toujours à l'adresse 0 de la mémoire, le compteur de programme étant le registre R(0). Il peut être nécessaire, en cours d'exécution de programme, d'attribuer le rôle de compteur de programme à un autre registre R (par exemple s'il faut libérer le registre R(0) de manière à pouvoir l'utiliser ultérieurement pour un accès direct à la mémoire). Le changement de compteur de programme est fait simplement en changeant par programme le contenu du registre P grâce à l'instruction SET P de mnémonique SEP.

Supposons que, à la ligne d'adresse Z de la mémoire, on ait inscrit le code D 4 de l'opération $4 \rightarrow P(\text{SEP } R(4))$. Lors du cycle de *recherche* de cette instruction, son code D 4 est chargé dans les registres I et N et le compteur de programme (registre R(0)) est incrémenté : son contenu devient $Z + 1$. Pendant le cycle d'*exécution* de SEP R(4), le contenu 4 du registre N est transféré dans le registre P. À l'issue de ce cycle, le *registre R(4) est compteur de programme*. Ceci signifie que, pendant le cycle de *recherche* de l'instruction suivante, c'est le contenu W du registre R(4) qui va être chargé dans le registre A et c'est la ligne d'adresse W qui va être pointée. C'est donc le code inscrit à l'adresse W qui va être chargé dans les registres I et N. C'est aussi le contenu du registre R(4) qui est incrémenté. Le programme se poursuivra donc, à partir de l'adresse W, le registre R(4) pointant les codes (et éventuellement les opérandes) des instructions successives à réaliser (voir fig. 3).

Il est bien entendu que le contenu initial W du registre R(4) aura dû être fixé préalablement par programme (voir également fig. 3).

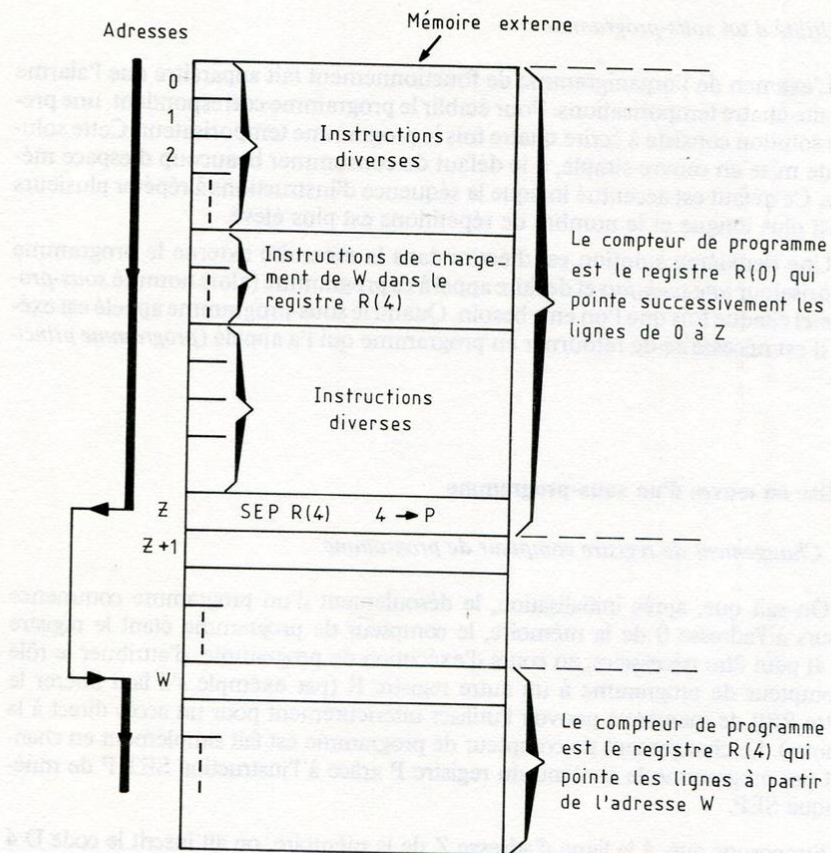


Fig. 3

Remarques

a) Cette opération réalise un *saut* de programme qu'il ne faut pas confondre avec une opération de branchement dans laquelle il n'y a pas de changement de compteur de programme mais uniquement une intervention sur le contenu du compteur de programme.

b) Sur la maquette d'essai (ensemble maquette A — maquette B), la mémoire ne comporte que 256 pas et est adressée seulement par les 8 bits de plus faible poids du registre R compteur de programme. Seul le contenu du demi-registre R(4). 0 doit donc être préalablement fixé.

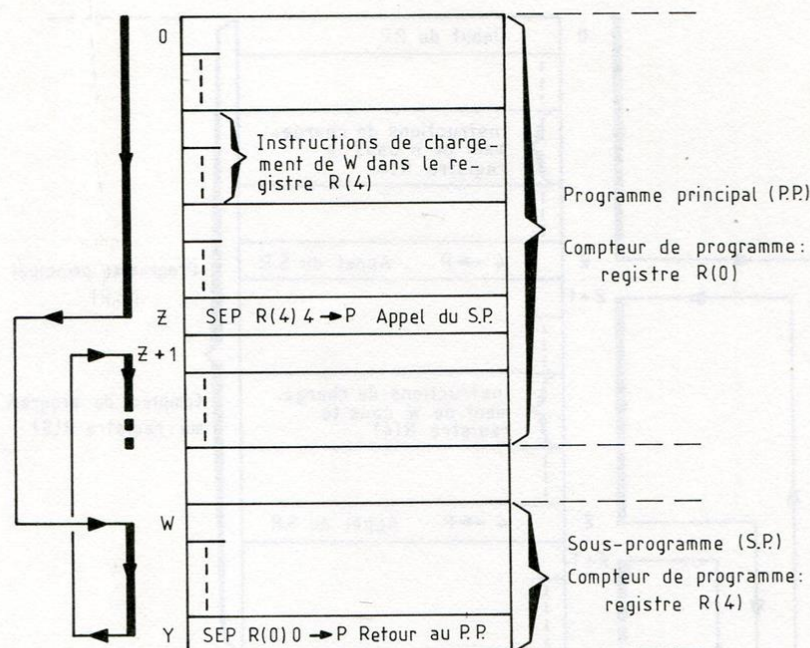


Fig. 4

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

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, retourner au programme principal.

Reportons-nous à la figure 4. Supposons que le sous-programme soit enregistré à partir de l'adresse W dans la mémoire et décidons que le registre R (4) sera compteur de programme pour ce sous-programme (on l'appellera donc compteur de sous-programme). Le programme principal est décrit en utilisant le registre R (0) comme compteur de programme. Il commence à l'adresse 0 et comporte entre autres l'initialisation du compteur de sous-programme, c'est-à-dire le chargement de l'adresse W du début du sous-programme dans le registre R (4).

Lors du déroulement du programme, on décrit d'abord le programme principal à partir de l'adresse 0. Lorsque l'on arrive à l'adresse Z, 4 est chargé dans le registre P et le compteur de programme devient donc le registre R (4). Le programme se poursuit par l'exécution de l'instruction dont le code est inscrit à l'adresse W préalablement chargée dans le registre R (4). L'exécution de l'instruction SEP R (4) constitue

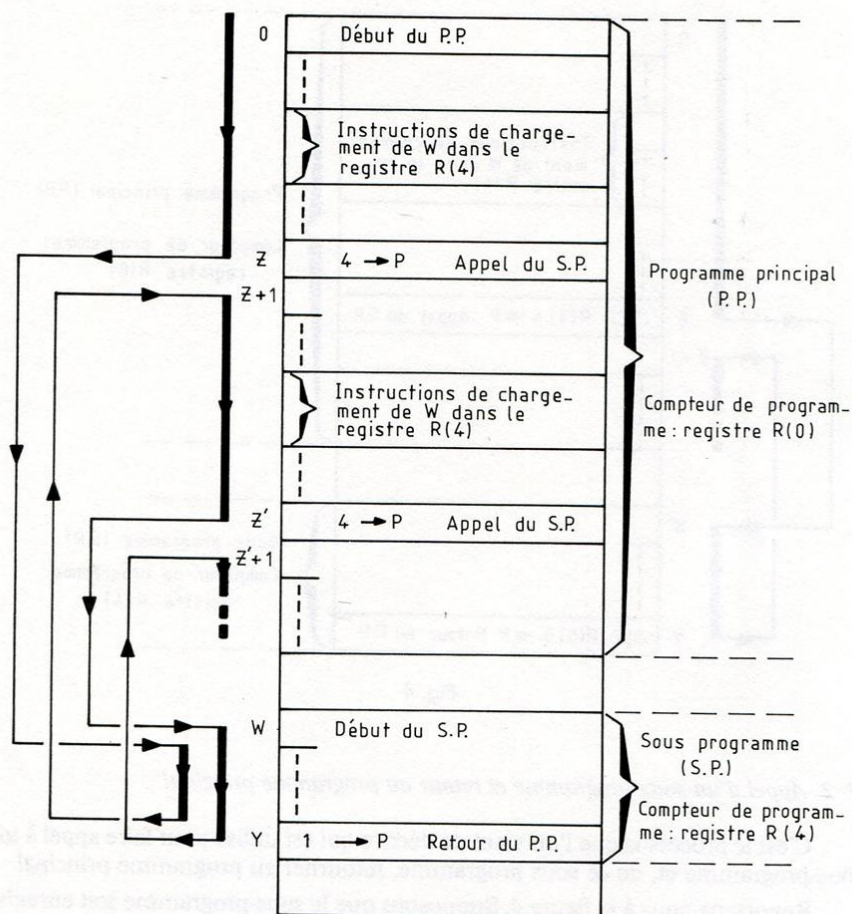


Fig. 5

donc l'appel du sous-programme. Le sous-programme enregistré à partir de l'adresse W s'exécute ensuite et se termine à l'adresse Y par l'instruction SEP R(0) qui a pour effet de confier à nouveau au registre R(0) la mission de compteur de programme. On se souvient que le registre R(0) avait été abandonné avec le contenu Z+1 et le programme principal peut donc se poursuivre, à partir de l'adresse Z+1. L'exécution, dans le sous-programme, de l'instruction SEP R(0) constitue donc l'ordre de retour au programme principal.

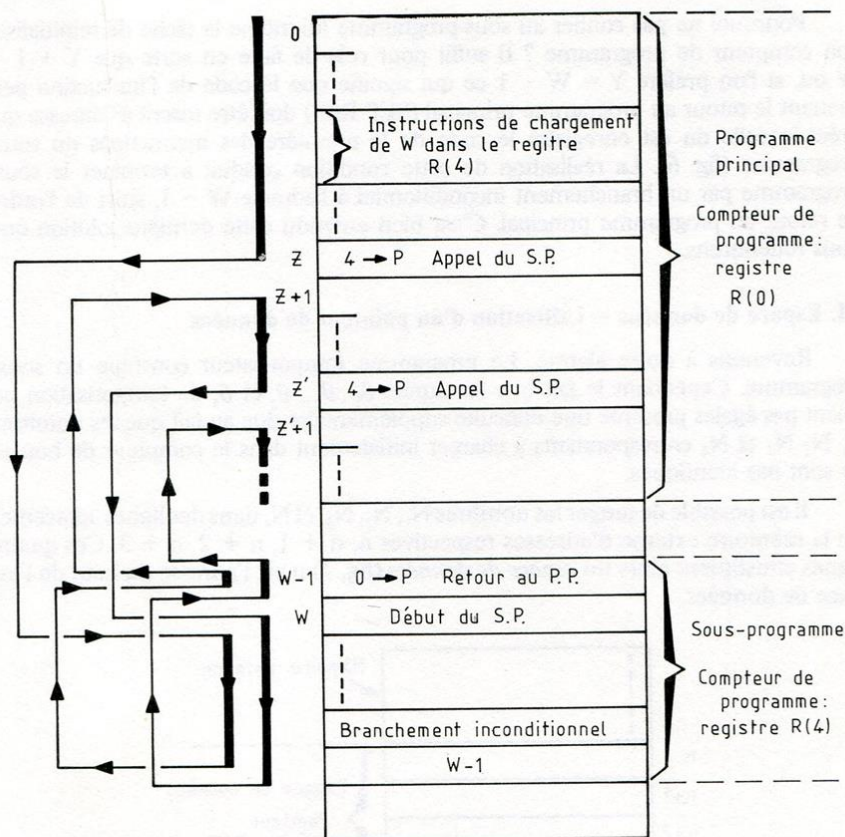


Fig. 6

Remarque très importante :

Lors du retour au programme principal, le registre R(4) est abandonné avec le contenu $Y + 1$. Or le programme principal va faire obligatoirement appel à nouveau au sous programme qui *début* à l'adresse W. Avant de procéder à un nouvel appel du sous-programme il est donc nécessaire de réinitialiser le compteur de sous-programme c'est-à-dire de charger à nouveau le registre R(4) par W. Pour deux appels du sous-programme, la structure du programme est alors celle représentée sur la figure 5. On remarque que, si l'on fait appel n fois au sous-programme, il faut initialiser n fois le registre R(4) compteur de sous-programme, ce qui nécessite une occupation importante de la mémoire si n est important.

Pourquoi ne pas confier au sous-programme lui-même la tâche de réinitialiser son compteur de programme ? Il suffit pour cela de faire en sorte que $Y + 1 = W$ ou, si l'on préfère $Y = W - 1$ ce qui signifie que le code de l'instruction permettant le retour au programme principal (SEP R(0)) doit être inscrit à l'adresse qui précède celle où est enregistré le code de la première des instructions du sous-programme (fig. 6). La réalisation de cette condition conduit à terminer le sous-programme par un branchement inconditionnel à l'adresse $W - 1$, suivi de l'ordre de retour au programme principal. C'est bien entendu cette dernière solution que nous retiendrons.

III. Espace de données – Utilisation d'un pointeur de données

Revenons à notre alarme. Le programme temporisateur constitue un sous-programme. Cependant le fait que les durées $\theta_1, \theta_2, \theta_3$ et θ_4 de temporisation ne soient pas égales présente une difficulté supplémentaire due au fait que les nombres N_1, N_2, N_3 et N_4 correspondants à charger initialement dans le compteur de boucle ne sont pas identiques.

Il est possible de ranger les nombres N_1, N_2, N_3 et N_4 dans des lignes adjacentes de la mémoire externe d'adresses respectives $n, n + 1, n + 2, n + 3$. Ces quatre lignes constituent alors un *espace de données* (fig. 7). n est l'adresse du haut de l'espace de données.

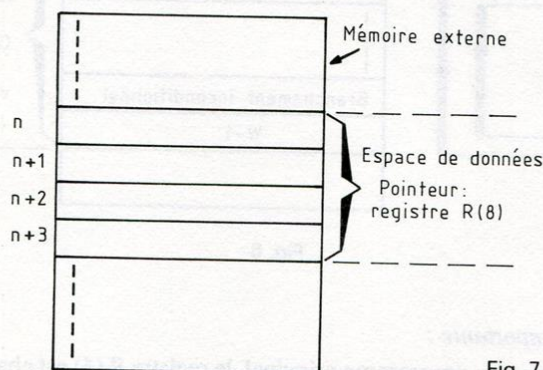


Fig. 7

Il faut que lors du premier appel du sous-programme temporisateur, N_1 soit chargé dans le compteur de boucle. L'examen du jeu d'instructions montre que ceci peut être réalisé à condition de confier à l'un des registres R le soin de pointer cette donnée. Soit, pourquoi pas, R(8) ce registre. L'instruction LOAD VIA N(LDN R(8)) qui réalise $M(R(8)) \rightarrow D$ permet de transférer N_1 dans le compteur de boucle (registre R(B) par exemple) à condition que R(8) ait été préalablement chargé par n , adresse où est inscrit N_1 .

Lors du deuxième appel du sous-programme, c'est N_2 qui doit être chargé dans le compteur de boucle. Or N_2 est enregistré à l'adresse $n + 1$. Il est donc nécessaire de procéder, avant le nouvel appel du sous-programme, à une incrémentation du registre R (8). Notez que la programmation de cette fonction peut être évitée en utilisant, à la place de LDN, l'instruction LOAD ADVANCE (LDA R(8)) qui en plus de $M(R(8)) \rightarrow D$ exécute précisément $R(8) + 1$, ce qui prépare le chargement de la donnée suivante. Le registre R (8) qui fournit ici les adresses des données inscrites dans l'espace des données et la mémoire est appelé *pointeur de données*.

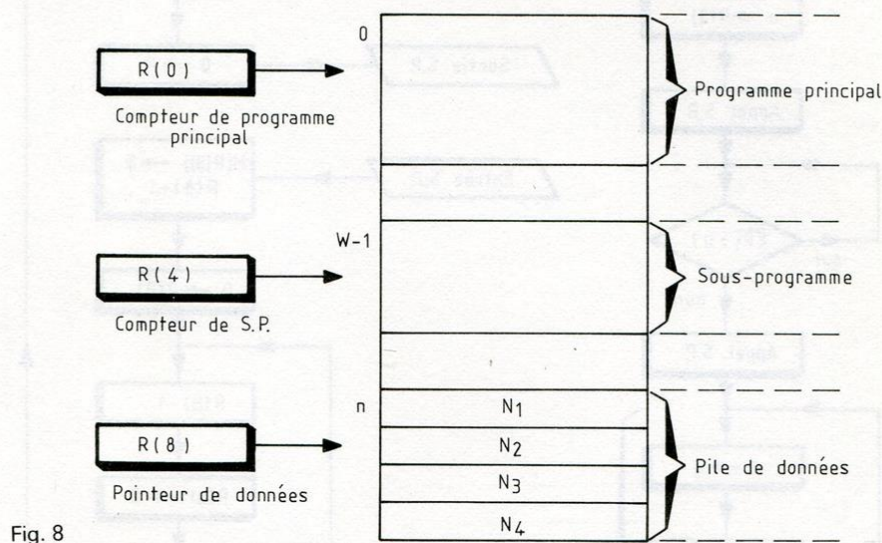


Fig. 8

IV. Elaboration du programme de l'alarme

IV.1. Organisation de l'espace mémoire disponible

L'espace mémoire doit être divisé en trois parties bien distinctes (fig. 8) :

- l'espace où est inscrit le programme principal, adressé par le compteur de programme principal (registre R(0)) et débutant donc à l'adresse 0.
- l'espace où est inscrit le sous-programme, adressé par le compteur du sous-programme (registre R(4) par exemple), débutant à l'adresse W et se terminant à l'adresse W - 1.

Ces deux espaces contiennent à la fois des *codes* d'instructions et des *opérandes* (données ou adresses).

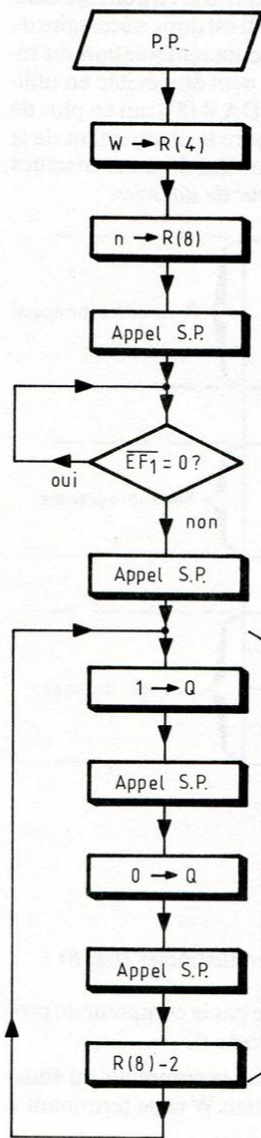


Fig. 9

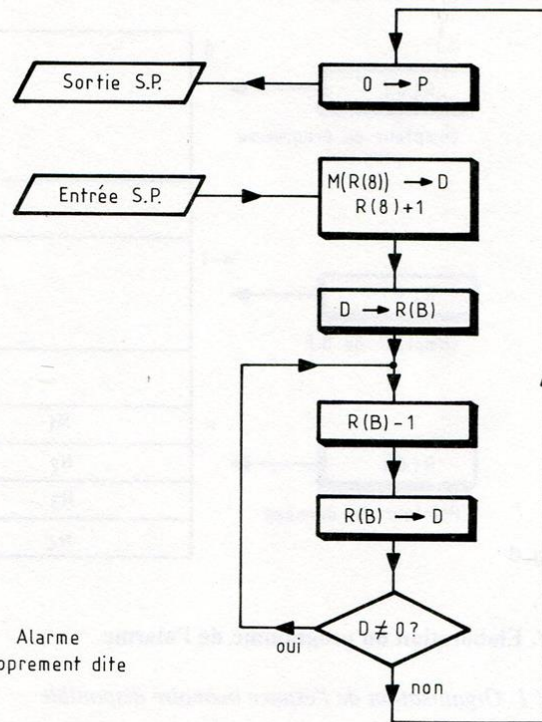


Fig. 10

— l'espace des données, adressé par le pointeur de données (registre R (8) par exemple).

Le registre R (B) sera arbitrairement choisi comme compteur de boucle, à décrémenter dans le sous-programme temporisateur.

IV. 2. Organigramme détaillé du programme principal (fig. 9)

On notera qu'il commence par l'initialisation du compteur de sous-programme (registre R (4)) et du pointeur de données (registre R (8)). Il faut aussi remarquer que, dans la partie alarme proprement dite correspondant à la commande de la sonnerie intermittente, il est nécessaire de décrémenter deux fois le registre R (8) pointeur de données avant de procéder au branchement. En effet, à l'issue de l'exécution du sous-programme temporisateur conduisant à la durée θ_4 , le contenu du registre R (8) est $n + 4$. Quand le branchement est exécuté, Q passe à 1 et l'on fait appel au sous-programme pour obtenir la durée θ_3 , correspondant à la donnée N_3 écrite à l'adresse $n + 2$. La double décrémentation ramène bien le contenu $n + 4$ du registre R (8) à $n + 2$.

IV. 3. Organigramme détaillé du sous-programme temporisateur (fig. 10)

Notez la mise en application de la technique de réinitialisation du compteur de sous-programme à la fin de l'exécution du sous-programme.

IV. 4. Ecriture du programme (fig. 11)

Les trois parties (programme principal, sous-programme, espace de données) occupent ici des espaces adjacents : ce n'est pas une obligation mais ceci facilite l'enregistrement du programme.

Au début de l'écriture du programme, on ne connaît pas forcément tous les opérandes. Ici par exemple on ne sait pas à quelle adresse se situera le début du sous-programme (W) ou le haut de l'espace de données (n). Il est recommandé dans ce cas d'inscrire, à la place des adresses ou données inconnues un signe quelconque (par exemple --) qui vous rappellera, en fin d'écriture, que vous avez un petit travail à achever. Nous l'avons fait sur la figure 11.

IV. 5. Vérification éventuelle

Rappelons que si N est grand, la durée de temporisation est $\theta = 48.N.T.$ Choisir une fréquence d'horloge adaptée et vérifier le bon déroulement du programme. La LED témoin de l'état de la bascule Q remplace évidemment la sonnerie. Le contact de porte est simulé par le commutateur de l'entrée de contrôle \overline{EFI} .

Adresse	Code	Opérande	Mnémonique	Commentaire
00	F8	1 3	LDI	} chargement adresse W du S.P dans R(4).0
02	A4		PLO R(4)	
03	F8	1 B	LDI	} chargement adresse n du haut de pile dans R(8).0
05	A8		PLO R(8)	
06	D4		SEP R(4)	4 → P : appel du S.P.
07	34	07	B1	Branch. à 07 si $\overline{EF}_1 = 0$; R(0) + 1 si $\overline{EF}_1 = 1$
09	D4		SEP R(4)	4 → P : appel du S.P.
0A	7B		SEQ	1 → Q
0B	D4		SEP R(4)	4 → P : appel du S.P.
0C	7A		REQ	0 → Q
0D	D4		SEP R(4)	4 → P : appel du S.P.
0E	28		DEC R(8)	} R(8) - 2
0F	28		DEC R(8)	
10	30	0A	BR	branch à 0A
12	D0		SEP R(0)	0 → P : retour au P.P
W→13	48		LDA R(8)	} chargement du compteur de boucle R(B)
14	AB		PLO R(B)	
15	2B		DEC R(B)	R(B) - 1
16	8B		GLO R(B)	R(B).0 → D
17	3A	15	BNZ	Branch. à 15 si D ≠ 0 ; R(4) + 1 si D = 0
19	30	12	BR	Branch. à 12
n→1B	N ₁			} Pile de données
1C	N ₂			
1D	N ₃			
1E	N ₄			

Fig. 11

Chapitre 14

ENTRÉE ET SORTIE DE DONNÉES

Dans tous les exemples que nous avons développés jusqu'ici, le système \mathcal{S} constitué du microprocesseur et de la mémoire préalablement chargée par un programme, ne fournit au monde extérieur le résultat de son travail que par l'intermédiaire de sa sortie Q. Par ailleurs, l'utilisateur peut intervenir dans le processus développé par le système \mathcal{S} par l'intermédiaire des quatre entrées notées \overline{EF}_1 à \overline{EF}_4 . Le système \mathcal{S} n'a donc été utilisé que comme un automate programmable à une sortie Q et quatre entrées de contrôle \overline{EF}_1 à \overline{EF}_4 .

Nous nous proposons d'utiliser les possibilités qu'offre l'ensemble maquette A + maquette B de porter à 12 le nombre des entrées (en utilisant *en cours de déroulement de programme* les 8 clés D_7 à D_0) et à 9 le nombre des sorties (grâce à la mise en œuvre d'un coupleur de sortie : voir à ce sujet le paragraphe VI du chapitre 7).

I. Entrée d'une donnée

I. 1. Position d'un problème

Nous avons, au chapitre 12, réalisé une bascule astable dont les durées de l'état haut et de l'état bas sont, à période d'horloge donnée, fixées par le choix des nombres W_1 et W_2 décrémentés au cours des programmes temporisateurs. Une modification de ces durées nécessite une modification des nombres W_1 et W_2 , donc un arrêt du déroulement du programme de manière à reprendre (partiellement) le chargement de la mémoire, ce qui n'est pas très commode.

Notre projet est ici de réaliser une bascule astable de rapport cyclique 1/2 (soit $W_1 = W_2$) dont la période est modifiable à tout moment, sans qu'il soit nécessaire d'arrêter le fonctionnement pour reprendre la programmation. La modification de la période d'oscillation des générateurs de laboratoire ne nécessite heureusement pas la mise à l'arrêt préalable !

I. 2. L'instruction d'entrée d'une donnée

La résolution de notre problème ne peut être effectuée que si l'on postule pour le système S la possibilité de « lire » constamment ou au moins périodiquement, la valeur commune W de W_1 et W_2 qui est affichée sur les clés D_7 à D_0 .

Ceci est réalisé grâce à l'utilisation de l'instruction INPUT (mnémonique INP). Lors du cycle d'exécution de cette instruction, ni la mémoire (car $\overline{MRD} = 1$) ni le microprocesseur n'imposent l'état du bus des données. La donnée présente sur les clés D_7 à D_0 est donc placée sur le bus. Cette donnée est chargée d'une part dans le registre D du microprocesseur, d'autre part dans la mémoire à l'adresse pointée par le registre R désigné par le registre X (registre R (X)). En notation symbolique, la fonction exécutée s'écrit : Donnée \rightarrow BUS ; BUS \rightarrow M (R (X)) et BUS \rightarrow D.

Remarque :

Le jeu d'instructions du COSMAC fait état de sept instructions d'entrées INP1 à INP 7. Ceci laisse entendre que la donnée à « entrer » peut parvenir sur le bus des données par sept voies différentes. La liaison de ces sept voies avec le bus des données se fait généralement par l'intermédiaire de sept coupleurs sélectionnés en utilisant les sorties N_2 , N_1 et N_0 du microprocesseur. Si par exemple on utilise l'instruction INP 5, lors du cycle d'exécution de cette instruction, $(N_2, N_1, N_0) = (1, 0, 1)$, c'est le coupleur de la voie 5 qui est sélectionné et c'est donc la donnée présentée par la voie 5 qui est chargée dans le registre D et dans la mémoire.

Les sorties N_2 , N_1 et N_0 recopient en fait les trois bits de plus faible poids du registre N, uniquement lors de l'exécution d'une instruction d'entrée ou de sortie de données. En dehors de ces phases, elles sont maintenues au niveau 0 (ce qui interdit l'utilisation de la combinaison $(N_2, N_1, N_0) = (0, 0, 0)$ pour sélectionner un coupleur).

Sur la maquette A, la liaison des clés D_7 à D_0 au bus des données se fait par l'intermédiaire de résistances : il s'agit en fait d'une huitième voie d'entrée de données qui offre la particularité de ne pas utiliser de coupleur et qui donc ne nécessite pas de validation. Le nombre binaire fixé par les clés se trouve sur le bus des données si ni le microprocesseur, ni la mémoire, ni l'un des sept coupleurs éventuels n'y a présenté une donnée. Comme aucune autre voie d'entrée n'est ménagée, c'est le nombre fourni par les clés qui se trouve sur le bus des données, quelle que soit la combinaison (N_2, N_1, N_0) , lors de l'exécution d'une instruction INP. En d'autres termes, n'importe laquelle des instructions INP 1 à INP 7 permet le chargement dans le registre D et dans la mémoire du nombre présenté par les clés D_7 à D_0 .

I. 3. Solution proposée

a) Elle correspond à l'organigramme de la figure 1 qui appelle les commentaires suivants :

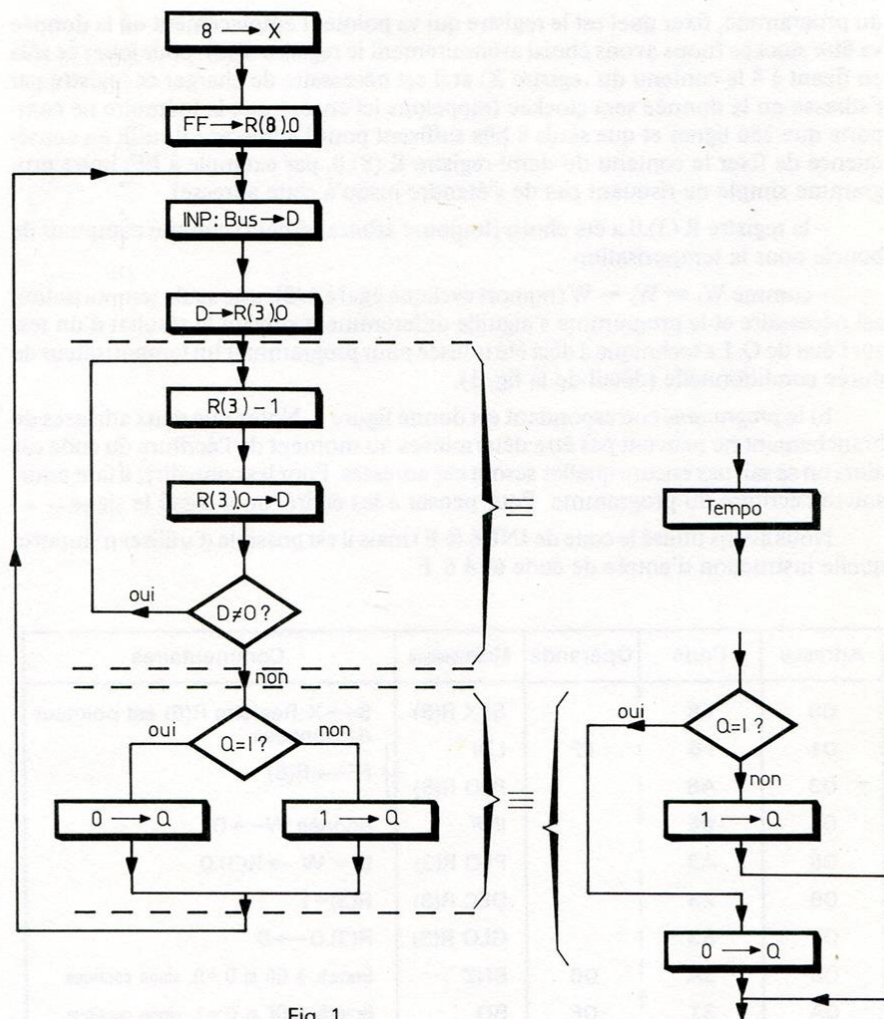


Fig. 1

– on utilise dans cette solution uniquement la fonction $BUS \rightarrow D$ de l'instruction INP.

– la fonction $BUS \rightarrow M(R(X))$ de la même instruction n'est pas utilisée mais elle est malgré tout exécutée ! En conséquence, il est *indispensable* de fixer l'adresse de la ligne de la mémoire où la donnée va être stockée si l'on ne veut pas risquer que ce stockage se fasse à une adresse utilisée par le programme. Il faut donc, dès le début

du programme, fixer quel est le registre qui va pointer l'emplacement où la donnée va être stockée (nous avons choisi arbitrairement le registre R(8) pour jouer ce rôle en fixant à 8 le contenu du registre X) et il est nécessaire de charger ce registre par l'adresse où la donnée sera stockée (rappelons ici encore que la mémoire ne comporte que 256 lignes et que seuls 8 bits suffisent pour l'adresser ; il suffit en conséquence de fixer le contenu du demi-registre R(8).0, par exemple à FF, notre programme simple ne risquant pas de s'étendre jusqu'à cette adresse)

– le registre R(3).0 a été choisi (toujours arbitrairement) comme compteur de boucle pour la temporisation.

– comme $W_1 = W_2 = W$ (rapport cyclique égal à 1/2), une seule temporisation est nécessaire et le programme s'aiguille différemment suivant le résultat d'un test sur l'état de Q. La technique a déjà été utilisée pour programmer un temporisateur de durée conditionnelle (détail de la fig. 1).

b) le programme correspondant est donné figure 2. Notez que deux adresses de branchement ne peuvent pas être déterminées au moment de l'écriture du code car alors on ne sait pas encore quelles seront ces adresses. Pour les connaître, il faut poursuivre l'écriture du programme. Pour penser à les écrire, on a laissé le signe –

Nous avons utilisé le code de INP 6 (6 E) mais il est possible d'utiliser n'importe quelle instruction d'entrée de code 69 à 6 F.

Adresse	Code	Opérande	Mnémonique	Commentaires
00	E8		SEX R(8)	8 → X: Registre R(8) est pointeur de données FF → R(8)
01	F8	FF	LDI	
03	A8		PLO R(8)	
04	6E		INP	donnée W → D
05	A3		PLO R(3)	D = W → R(3).0
06	23		DEC R(3)	R(3) - 1
07	83		GLO R(3)	R(3).0 → D
08	3A	06	BNZ	Branch. à 06 si D ≠ 0, sinon continue
0A	31	0E	BQ	Branch. à 0F si Q = 1, sinon continue
0C	7B		SEQ	1 → Q
0D	30	10	BR	Branch. à 10
0F	7A		REQ	0 → Q
10	30	04	BR	Branch. à 04

Fig. 2

c) Si vous possédez la maquette A, enregistrez votre programme (ou celui que nous avons établi ensemble) et essayez-le en choisissant une fréquence d'horloge suffisamment faible et/ou un nombre W suffisamment grand pour que vous puissiez contrôler la période de votre bascule (sensiblement $2.48 W.T.$ si $W \gg 1$).

La période la plus longue est obtenue pour $W = 0$ (en effet la première décrémentation du registre $R(3)$ amène $R(3).0$ à 255), la plus courte pour $W = 1$. Amusez-vous à modifier la période en agissant sur les clés.

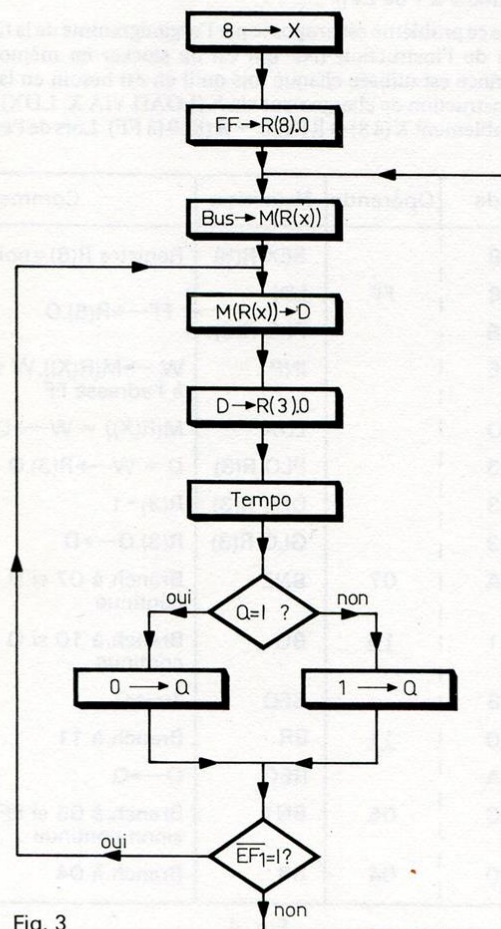


Fig. 3

1. 4. Variante

On souhaite maintenant que la modification de la période de la bascule ne se fasse pas forcément lors de la manipulation des clés D_7 à D_0 . On voudrait en somme *avoir la possibilité de programmer* sur les clés un nombre W qui fixera la période d'oscillation seulement lorsque l'on manœuvrera un commutateur K .

Ce commutateur K peut être la clé \overline{EF}_1 et une autre formulation du problème est : si $\overline{EF}_1 = 0$, la période de la bascule « suit » les indications des clés ; si $\overline{EF}_1 = 1$, une action sur les clés est sans effet sur la période de la bascule.

On conçoit la nécessité d'une mémorisation du nombre W présenté sur les clés D_7 à D_0 juste avant la transition 0 à 1 de \overline{EF}_1 .

Une solution de ce problème est proposée par l'organigramme de la figure 3. On utilise ici la seconde fonction de l'instruction INP qui est de stocker en mémoire la donnée W à « entrer ». Cette donnée est utilisée chaque fois qu'il en est besoin en la transférant dans le registre D grâce à l'instruction de chargement via X (LOAD VIA X , LDX) comme on a fixé préalablement X (à 8) et $R(X).0 = R(8).0$ (à FF). Lors de l'exécution de LDX, le

Adresse	Code	Opérande	Mnémonique	Commentaire
00	E8	FF	SEX R(8)	Registre R(8)=pointeur de donnée
01	F8		LDI	} FF → R(8).O
03	A8		PLO R(8)	
04	6E		INP	W → M(R(X)). W stocké à l'adresse FF
05	FO		LDX	M(R(X)) = W → D
06	A3		PLO R(3)	D = W → R(3).O
07	23		DEC R(3)	R(3) - 1
08	83		GLO R(3)	R(3).O → D
09	3A	07	BNZ	Branch. à 07 si D ≠ 0, sinon continue
0B	31	10	BQ	Branch. à 10 si Q = 1 sinon continue
0D	7B		SEQ	1 → Q
0E	30	11	BR	Branch. à 11
10	7A		REQ	0 → Q
11	3C	05	BN1	Branch. à 05 si $\overline{EF}_1 = 0$ ou $\overline{EF}_1 = 1$ sinon continue
13	30	04	BR	Branch. à 04

Fig. 4

contenu W de la ligne mémoire d'adresse FF est transféré dans le registre D. Grâce à un branchement conditionné à l'état de \overline{EF}_1 , on s'arrange pour que la modification du contenu W de la ligne d'adresse FF ne se fasse (par l'intermédiaire de l'instruction INP) que lorsque $\overline{EF}_1 = 0$.

Etablissez le programme correspondant. Il est donné figure 4. A votre maquette !

II. Sortie d'une donnée

II. 1. Position d'un problème

Il est possible de connaître à tout moment la période de la bascule astable programmée au paragraphe I. 3. du présent chapitre. Il suffit pour cela de convertir en décimal le nombre W codé en binaire sur les clés D_7 à D_0 et d'appliquer la relation approchée : Période $\approx 2.48 \cdot W.T$. La position des clés constitue en quelque sorte un affichage – certes précaire – de la période. Ce n'est plus le cas dans la variante proposée au paragraphe I. 4. car lorsque $\overline{EF}_1 = 1$ l'indication des clés ne correspond pas obligatoirement avec la période de la bascule. La position des clés peut avoir été modifiée sans que la période le soit.

On se propose d'imaginer un dispositif permettant l'affichage permanent du nombre W auquel la période est proportionnelle (si $W \gg 1$), et qui n'est pas obligatoirement celui qui est présent sur les clés.

II. 2. L'instruction de sortie d'une donnée

Rappelons que la maquette B comporte un registre latch dont les 8 entrées sont reliées au bus des données et dont l'état des 8 sorties est affiché en permanence grâce à 8 LED ou deux afficheurs hexadécimaux. Le registre latch est transparent lorsque son entrée de commande CL (reliée à la sortie TPB) du microprocesseur est au niveau 1 et ceci à condition que son entrée de contrôle CS (reliée à la sortie N_0 du microprocesseur) soit aussi à 1. La donnée ainsi transférée en sortie est mémorisée au moment du front descendant de TPB.

La sortie d'une donnée et son affichage (binaire ou hexadécimal) se fait en utilisant une des instructions OUTPUT (OUT) qui sélectionne le registre latch coupleur, c'est-à-dire qui provoque le passage de N_0 à l'état 1 (OUT 1 par exemple, mais aussi OUT 3, OUT 5 et OUT 7). L'exécution de cette instruction place la mémoire en position lecture en l'adressant par le registre R désigné par le registre X (registre R(X)) : le mot $M(R(X))$ apparaît donc sur le bus des données. Le signal de commande $CL = TPB$ validé par $CS = N_0 = 1$ provoque le chargement de $M(R(X))$ sur le registre coupleur et donc conduit à l'affichage de $M(R(X))$. De plus, l'exécution de cette instruction incrémente le registre R(X) pointeur. En notation symbolique, les opérations effectuées pendant l'exécution des instructions OUT 1, OUT 3, OUT 5 et OUT 7 sont sur notre maquette : $M(R(X)) \rightarrow BUS \rightarrow \text{afficheurs}; R(X) + 1$. Bien entendu, les instructions OUT 2, OUT 4 et OUT 6 qui laissent N_0 à 0 permettent

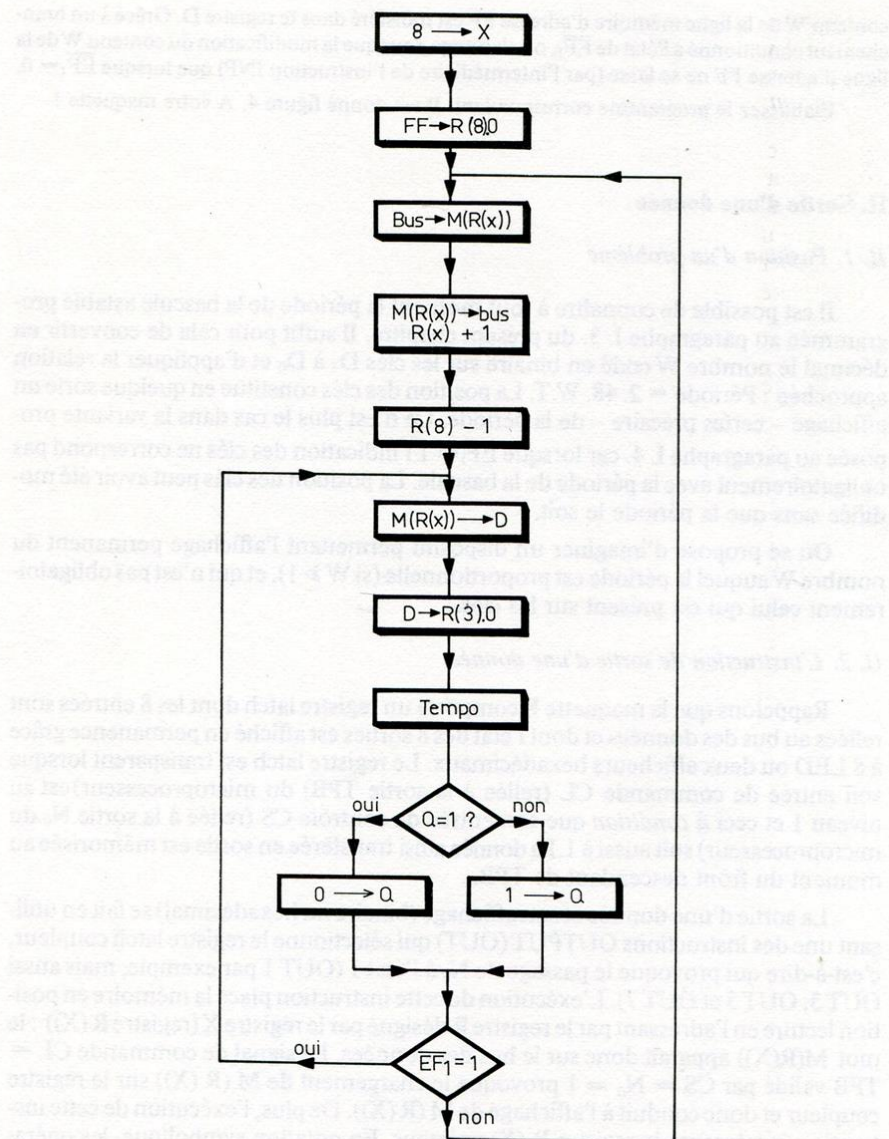


Fig. 5

$M(R(X)) \rightarrow \text{BUS}$ et $R(X) + 1$ sans valider CL et donc sans permettre la sortie de $M(R(X))$ vers les afficheurs.

II. 3. Solution proposée

Elle consiste en une légère modification de l'organigramme de la figure 3 conduisant à l'organigramme de la figure 5. Dès que le nombre W correspondant à la période de l'astable est « entré » (c'est-à-dire en particulier stocké à l'adresse R(X)) intervient une instruction de sortie : le nombre W inscrit à l'adresse R(X) est donc immédiatement « lu », mémorisé sur le coupleur et affiché. On notera la présence d'une décrémentation du registre $R(X) = R(8)$ juste après la « sortie », destinée à compenser l'incrémentation ici inutile intervenant lors de l'exécution de l'instruction OUT.

Le programme de la figure 6 reproduit pratiquement celui de la figure 4 : seuls les contenus des lignes 05 et 06 (codes des instructions OUT 1 et DEC R(8)) sont ajoutés ; les contenus des lignes suivantes sont identiques à un décalage de deux

Adresse	Code et opérande	Mnémonique
00	E8	SEX R(8)
01	F8 FF	LDI
03	A8	PLO R(8)
04	6E	INP
05	61	OUT 1
06	28	DEC R(8)
07	FO	LDX
08	A3	PLO R(3)
09	23	DEC R(3)
0A	83	GLO R(3)
0B	3A 09	BNZ
0D	31 12	BQ
0F	7B	SEQ
10	30 13	BR
12	7A	REQ
13	3C 07	BN1
15	30 04	BR

Fig. 6

unités d'adresses près (attention aux modifications éventuelles à apporter aux opérantes adresses).

Remarque

Pour résoudre le problème posé, nous avons utilisé cette méthode dans le seul but d'utiliser (afin de la connaître) l'instruction OUT. En fait, il n'était pas *ici* indispensable d'utiliser cette instruction pour afficher le nombre W proportionnel à la période de l'astable. L'organigramme montre clairement que l'instruction OUT 1 ramène sur le bus la donnée W qui y était présente au moment de l'exécution de INP. Le choix de OUT 1 permet en plus de sélectionner le coupleur de sortie grâce à l'apparition de $CS = N_0 = 1$.

Notons que cette dernière fonction est aussi réalisée par INP 1, 3, 5 et 7. Si au lieu de choisir INP 6 (code 6 E) pour instruction d'entrée comme nous l'avons fait dans le programme de la figure 4 nous avions choisi par exemple INP 5 (code 6 D), au moment où W est présent sur le bus des données pour être chargé dans le registre D et la mémoire, le coupleur de sortie est activé car $N_0 = 1$ et W apparaît sur les afficheurs de sortie sans qu'il soit nécessaire d'utiliser l'instruction OUT. Faites l'essai !

III. Exemples de programmation

Les exemples traités ici ont un double but :

- familiarisation avec l'emploi des instructions d'entrée et de sortie
- utilisation d'instructions qui n'ont pas été rencontrées jusque là.

Les solutions que nous proposons ne sont certainement pas uniques. Pour chaque exercice, il est intéressant d'établir son propre programme, éventuellement de l'essayer, et de comparer à la solution proposée.

III. 1. Réalisation d'une sorte de « chenillard »

a) en utilisant une instruction de décalage circulaire

On désire allumer les LED d'affichage binaire d'une donnée dans l'ordre indiqué par la figure 7.

Nous utilisons pour cela l'instruction RING SHIFT RIGHT (décalage circulaire à droite) de mnémonique R S H R dont la fonction est résumée par la figure 8.

Le contenu du registre D est décalé à droite. Le bit de poids le plus faible est chargé dans le registre DF. Le contenu initial du registre DF est mis à la place du bit de poids fort du registre D. La solution proposée (organigramme de la fig. 9) consiste à entrer d'abord le nombre binaire 11000000 (soit CO en hexadécimal) affiché sur les clés D_7 à D_0 sur le registre D et à l'adresse FO de la mémoire grâce à l'instruction INP. On prend soin de désigner préalablement le pointeur (ici registre R(8)) et d'en fixer le contenu de sa partie de faible poids R(8).0 (ici à FO). L'instruction OUT permet la sortie du nombre CO stocké à l'adresse FO et son affichage : les LED D_7 et D_0 s'allument. Une décrémentation du registre R(8) est destinée à compenser l'incrément-

	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	●	●	○	○	○	○	○	○
2	○	●	●	○	○	○	○	○
3	○	○	●	●	○	○	○	○
4	○	○	○	●	●	○	○	○
5	○	○	○	○	●	●	○	○
6	○	○	○	○	○	●	●	○
7	○	○	○	○	○	○	●	●
8	○	○	○	○	○	○	○	●
9	●	○	○	○	○	○	○	○
10	●	●	○	○	○	○	○	○
11	○	●	●	○	○	○	○	○

etc.

Fig. 7

tation survenue lors de l'exécution de OUT et ramène le contenu de R(8).0 à FO. Le premier décalage circulaire à droite réalisé amène à 0110000 le contenu du registre D. Ce nombre binaire est chargé à l'adresse R(8).0 = FO de la mémoire en utilisant l'instruction STR déjà rencontrée. Il peut alors être sorti et affiché à son tour en revenant à l'instruction OUT. Cette séquence d'opérations se reproduit indéfiniment.

Le programme correspondant est écrit sur la figure 10.

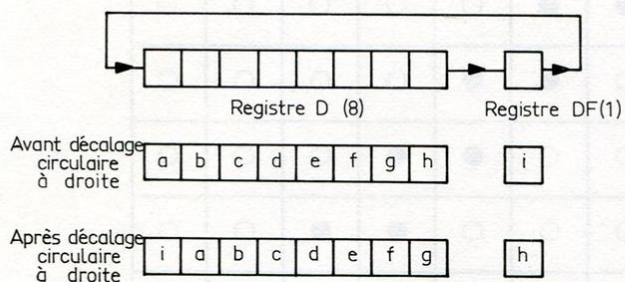


Fig. 8

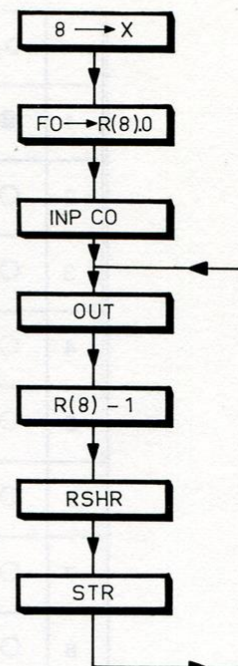


Fig. 9

Adresse	Code et opérande	Mnémonique	Commentaire
00	E8	SEX R(8)	8 → X : Registre R(8) pointeur FO → R(8).0
01	F8 FO	LDI	
03	A8	PLO R(8)	
04	6A	INP 2	CO → D et CO → M(R(8)) = M(FO)
05	61	OUT 1	M(R(8)) → afficheurs; R(8) + 1
06	28	DEC R(8)	R(8) - 1
07	76	RSHR	décalage circulaire à droite
08	58	STR	D → M(R(8)) = M(FO)
09	30 05	BR	Branch à 05

Fig. 10

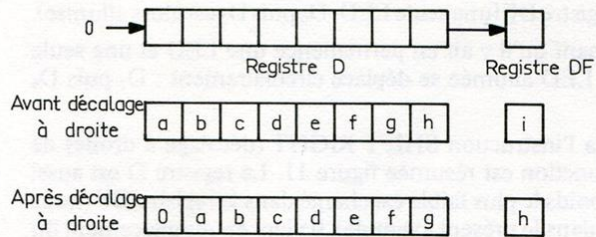


Fig. 11

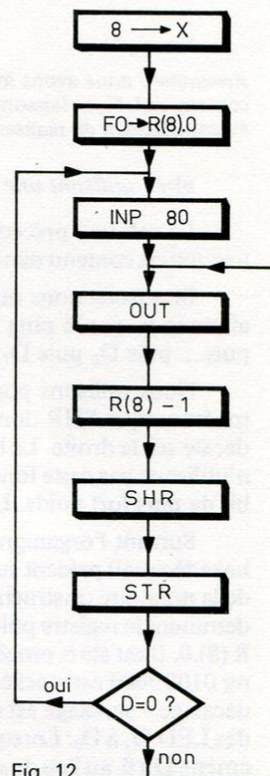


Fig. 12

Adresse	Code et opérande	Mnémonique	Commentaire
00	E8	SEX R(8)	
01	F8 F0	LDI	
03	A8	PLO R(8)	
04	6A	INP 2	80 → D et 80 → M(R(8)) = M(F0)
05	61	OUT 1	M(R(8)) = M(F0) → afficheurs
06	28	DEC R(8)	
07	F6	SHR	décalage à droite
08	58	STR	D → M(R(8))
09	32 04	BZ	branch à 04 si D=0, sinon continue
0B	30 05	BR	branch à 05

Fig. 13

Remarque : nous avons implicitement supposé que le registre DF présente initialement un contenu nul. Nous laissons au lecteur le soin d'imaginer un programme préalable permettant éventuellement de réaliser cette condition.

b) en utilisant une instruction de décalage (non circulaire)

La solution précédente conduit à allumer deux LED sauf dans le cas où un 1 logique est contenu dans le registre DF (une seule LED, D_0 puis D_7 est alors allumée).

Imposons nous maintenant qu'il y ait en permanence une LED et une seule allumée et que le rang de la LED allumée se déplace circulairement : D_7 puis D_6 puis... puis D_0 puis D_7 ...

Nous utilisons pour cela l'instruction SHIFT RIGHT (décalage à droite) de mnémonique SHR dont la fonction est résumée figure 11. Le registre D est aussi décalé sur la droite. Le bit le poids le plus faible est chargé dans le registre DF (nous n'utilisons pas cette fonction dans le présent exemple). 0 vient en remplacement du bit de plus fort poids. Le contenu initial du registre DF est perdu.

Suivant l'organigramme de la figure 12, le nombre binaire 10000000 (80 en hexadécimal) présent sur les clés D_7 à D_0 est chargé sur le registre D et à l'adresse FO de la mémoire (instruction INP) puis ressorti et affiché grâce à OUT. Comme précédemment le registre pointeur R (8) est décrémenté pour ramener à FO le contenu de R (8).0. Il est alors procédé à un décalage à droite du registre D. Son nouveau contenu 0100 0000 est stocké à l'adresse FO contenue dans R (8).0. La séquence sortie – décalage – stockage est reprise jusqu'à ce que $D = 0$ et entraîne l'allumage successif des LED D_7 à D_0 . Lorsque la dernière LED D_0 est allumée, le décalage à droite de D amène D à 0, au lieu de sortir directement la donnée (ce qui conduirait à éteindre toutes les LED), un branchement conditionnel permet de revenir à l'entrée du nombre 80 qui, sorti et affiché, conduit à allumer la diode D_7 . Le cheminement reprend... On note que la donnée 80 doit être maintenue sur les clés puisque le programme y fait appel périodiquement.

c) Si vous en avez la possibilité, essayez ces programmes en partant d'une période d'horloge suffisamment grande pour bien observer le déroulement des opérations successives puis diminuez progressivement cette période jusqu'à obtenir l'effet habituel de chenillard

III. 2. Opérations logiques sur deux octets

a) énoncé du problème

On désire effectuer une opération logique (ou, ou exclusif, et) sur deux octets W_1 et W_2 avec les conditions suivantes :

– l'octet W_1 est inscrit dans la mémoire (à une place fixée par le programmeur)

– l'octet W_2 est présent sur les clés de programmation dès le lancement du déroulement du programme

– le résultat $W_1 \circ W_2$ doit être sorti et affiché (sur les LED et les afficheurs hexadécimaux)

b) solution proposée

Il faut d'abord savoir que les différentes opérations logiques se font entre bits correspondants des octets W_1 et W_2 . Si $W_1 = a_1 b_1 c_1 d_1 e_1 f_1 g_1 h_1$ et $W_2 = a_2 b_2 c_2 d_2 e_2 f_2 g_2 h_2$ alors $W_1 \circ W_2 = a_1 \circ a_2 b_1 \circ b_2 \dots h_1 \circ h_2$

Exemple numérique

$W_1 = 11001011$

soit CB en hexadécimal

$W_2 = 10110001$

soit B1 en hexadécimal

L'opération logique « ou » conduit à :

$W_1 \vee W_2 = 11111011$

soit FB en hexadécimal.

L'opération logique « ou exclusif » conduit à :

$W_1 \oplus W_2 = 01111010$

soit 7A en hexadécimal

L'opération logique « et » conduit à :

$W_1 \odot W_2 = 10000001$

soit 81 en hexadécimal

L'examen du jeu d'instructions montre que chacune des opérations logiques peut être réalisée entre le contenu du registre D et une donnée inscrite en mémoire à une adresse qui peut être fournie soit par le registre pointeur R(X) (il faut alors que X soit fixé ainsi que le contenu du registre R(X)) soit par le registre compteur de programme R(P) (dans ce cas, la donnée est écrite à l'adresse qui suit le code de l'opération : on a affaire à un adressage immédiat).

Nous proposons d'utiliser ici l'adressage immédiat (organigramme fig. 14).

L'instruction INP permet de charger le registre D par le nombre W_2 affiché sur les clés. (Il ne faut toujours pas oublier que W_2 est aussi stocké en mémoire à l'adresse R(X) d'où la nécessité de fixer préalablement X et R(X).0). L'opération immédiate (ou immédiat, ou exclusif immédiat ou bien et immédiat) effectue $W_1 \circ W_2$ et amène le résultat sur le registre D. L'instruction de stockage STR permet de transférer le résultat $W_1 \circ W_2$ à l'adresse R(X).0 = R(8).0 = FO en vue de sa sortie et de son affichage sur les LED et les afficheurs hexadécimaux.

Ceci conduit au programme de la figure 15 où FY désigne le code de l'opération logique immédiate effectuée (F9 pour OR IMMEDIATE, FB pour EXCLUSIVE OR IMMEDIATE, FA pour AND IMMEDIATE de mnémoniques respectives ORI, XRI, ANI).

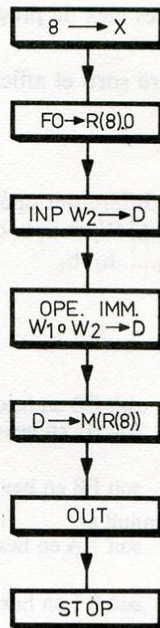


Fig. 14

Adresse	Code et opérande	Mnémonique	Commentaire
00	E8	SEX R(8)	
01	F8 FO	LDI	
03	A8	PLO R(8)	
04	6A	INP 2	$W_2 \rightarrow D$ et $W_2 \rightarrow M(R(8))$
05	FY W_1	OPI	$W_1 \circ W_2 \rightarrow D, R(0) + 1$
07	58	STR	$W_1 \circ W_2 = D \rightarrow M(R(8))$
08	61	OUT 1	$W_1 \circ W_2 = M(R(8)) \rightarrow \text{afficheurs}$
09	00	IDL	attente

FY : code de l'opération immédiate (OPI)

Fig. 15

III. 3. Opérations arithmétiques sur deux octets

a) énoncé du problème

On désire effectuer une opération arithmétique (addition ou soustraction) sur deux octets W_1 et W_2 (W_1 en mémoire, W_2 présent sur les clés) et afficher le résultat.

Une difficulté supplémentaire intervient par le fait que le résultat est un nombre de 9 bits. Les 8 bits de plus faible poids sont chargés sur le registre D alors que le registre DF reçoit le bit de poids le plus fort. On s'impose que les bits de plus faible poids du résultat soient transférés sur les afficheurs de sortie et que la LED témoin de la bascule Q indique le neuvième bit de poids le plus fort.

b) solution proposée

Pour changer un peu, nous proposons de placer au cours de la programmation W_1 à l'adresse 02 de la mémoire et nous réservons la ligne d'adresse 01 pour recevoir en cours de programme, W_2 puis l'octet correspondant aux 8 bits de plus faible poids du résultat $W_1 \circ W_2$. Cette contrainte va nous obliger à utiliser le registre R (X) comme pointeur de donnée au lieu d'utiliser le registre R (P) compteur de programme à cet effet.

L'organigramme de la figure 16 indique que l'on commence par réaliser un saut long (LONG SKIP : LSKP). Le code C8 de cette instruction étant inscrit à l'adresse 00 de la mémoire, le programme se poursuit par l'exécution de l'instruction dont le code est écrit à l'adresse 03 : les adresses 01 et 02 sont ignorées. Le registre R (5) - pourquoi pas ? - est ensuite choisi comme pointeur de données. Le contenu de R(5).0 est fixé à 01 de sorte que l'instruction INP qui suit permet de charger W_1 à l'adresse 01 de la mémoire (où elle ne gêne pas) et dans le registre D. Une incrémentation du registre R (5) amène R (5) . 0 à 02. L'une des opérations (voir liste des instructions) peut être effectuée : ADD (soit $W_1 + W_2$ car $M(R(X)) = M(R(5).0) = W_1$ et $D = W_2$) ou SUBTRACT D (soit $W_1 - W_2$) ou SUBTRACT MEMORY (soit $W_2 - W_1$). Le registre R (5) est décrémenté ce qui ramène R (5).0 à 01, adresse où l'on va stocker les huit bits de poids faible du résultat contenus dans le registre D, grâce à l'instruction STR. Ces huit bits, sont alors sortis et affichés sur les LED (instruction OUT). Il reste à « sortir » le contenu du registre DF. A cet effet l'état du registre est testé. Si DF = 0, un branchement s'effectue et conduit à l'arrêt du séquençement (en effet Q étant mis à 0 lors de l'initialisation, son état est conforme à l'état de DF et il n'y a pas de modification à apporter). Si DF = 1, le programme se poursuit par la mise à 1 de Q (pour recopier DF) puis l'arrêt du séquençement. Le programme est fourni figure 17.

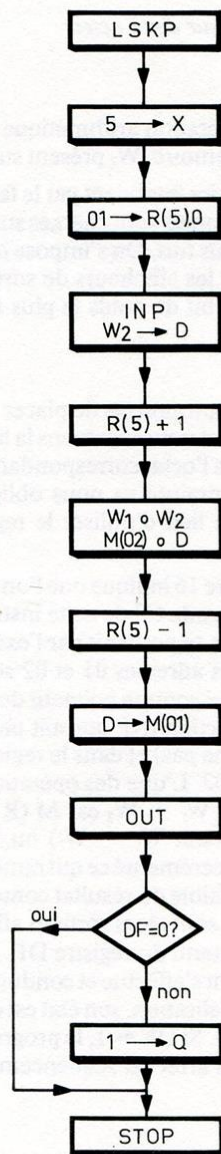


Fig. 16

Adresse	Code et opérande	Mnémonique	Commentaire
00	C8 00 W ₁	LSKP	
03	E5	SEX R(5)	5 → X
04	F8 01	LDI	} 01 → R(5).0
06	A5	PLO R(5)	
07	6A	INP 2	
08	15	INC R(5)	
09	FY	OPE	W ₁ o W ₂ → DF, D
0A	25	DEC R(5)	
0B	55	STR	D → M (R(5)) = M(01)
0C	61	OUT 1	M(01) → afficheurs
0D	3B 10	BNF	Branch. si DF = 0
0F	7B	SEQ	1 → Q
10	00	IDL	attente

Le code FY de l'opération à effectuer (OPE) est :

F4 pour ADD (ADD)

F5 pour SUBTRACT D (SD)

F7 pour SUBTRACT MEMORY (SM)

Fig. 17

c) vérification et interprétation des résultats.

Cas de l'addition

* exemple 1 :

$$\begin{array}{r}
 W_1 = 01001011 \quad 4B \\
 + W_2 = 10011000 \quad \text{ou} \quad + 98 \\
 \hline
 W_1 + W_2 = 11100011 \quad E3
 \end{array}$$

W₁ + W₂ est un nombre binaire de format 8 bits (W₁ + W₂ < 256 en décimal ou 100 en hexadécimal). Ces huit bits sont inscrits sur le registre D et 0 est chargé dans le registre DF.

* exemple 2 :

$$\begin{array}{r}
 W_1 = 11001011 \quad CB \\
 + W_2 = 10110001 \quad \text{ou} \quad + B1 \\
 \hline
 W_1 + W_2 = 101111100 \quad 17C
 \end{array}$$

W₁ + W₂ est un nombre binaire de format 9 bits (W₁ + W₂ ≥ 256 en décimal ou 100 en hexadécimal). Les 8 bits de droite sont inscrits sur le registre D et le bit de poids le plus fort 1 est chargé sur le registre DF.

Cas de la soustraction

Au lieu d'effectuer la soustraction $W_1 - W_2$, l'ALU effectue l'addition de W_1 et du complément à deux de W_2 : le résultat peut donc comporter 9 bits, son interprétation dépend de la valeur du bit de poids le plus fort chargé dans le registre DF.

* exemple 1

$$\begin{array}{rcl} W_1 & = & 11001011 \quad \text{CB} \\ - W_2 & = & 10110001 \quad \text{ou } - \text{B1} \\ \hline W_1 - W_2 & = & 00011010 \quad \text{1A} \end{array}$$

$W_1 > W_2$, le résultat $W_1 - W_2$ est positif. L'ALU procède de la manière suivante :

$$\begin{array}{rcl} 11001011 & \rightarrow & W_1 \\ + 01001110 & \rightarrow & \text{complément logique de } W_2 \\ + 1 & & \text{complément à deux de } W_2 \\ \hline 100011010 & \rightarrow & \text{chargé dans le registre D (= 1A en hexa.)} \\ & \rightarrow & \text{chargé dans le registre DF.} \end{array}$$

Le bon résultat est dans le registre D : le 1 chargé dans le registre DF l'atteste.

* exemple 2

$$\begin{array}{rcl} W_2 & = & 10110001 \quad \text{B1} \\ - W_1 & = & 11001011 \quad \text{ou } - \text{CB} \\ \hline W_2 - W_1 & = & ? \quad - \text{1A} \end{array}$$

Le résultat est évidemment négatif. En binaire, un nombre négatif est écrit en utilisant la convention de complémentation à 2.

Suivons le procédé développé par l'ALU :

$$\begin{array}{rcl} 10110001 & \rightarrow & W_2 \\ + 00110100 & \rightarrow & \text{complément à deux de } W_1 \\ + 1 & & \\ \hline 011100110 & \rightarrow & \text{chargé dans le registre D (= E6 en hexadécimal)} \\ & \rightarrow & \text{chargé dans le registre DF.} \end{array}$$

Le 0 chargé dans le registre DF indique que le résultat est négatif et est contenu dans le registre D sous forme du complément à 2.

Le complément à 2 de 11100110 est $00011001 + 1 = 00011010$ soit 1A en hexadécimal.

Conclusion :

Dans le cas de l'addition de deux octets, le registre DF contient le neuvième bit du résultat (poids le plus fort).

Dans le cas de la soustraction, si $DF = 1$, le résultat est positif et est contenu dans le registre D.

Si $DF = 0$, le résultat est négatif et est contenu dans le registre D sous forme du complément à 2.

Chapitre 15

L'INTERRUPTION PAR LE CANAL $\overline{\text{DMA-IN}}$

I. Position d'un problème et éléments de résolution.

Vous avez remarqué que le calculateur de la fin du chapitre précédent, destiné à effectuer une opération logique ou arithmétique sur deux octets, n'est pas commode à utiliser. En effet, l'un des deux nombres (W_1) doit être chargé dans la mémoire lors du chargement du programme. Une modification éventuelle de ce nombre oblige à reprendre la programmation comme on le fait lorsqu'une erreur est commise (utilisation du mode de fonctionnement LOAD et du commutateur RAM-ROM : voir plus haut).

Nous voulons maintenant perfectionner le système de manière à pouvoir entrer le nombre W_1 en mémoire, non plus *avant* le déroulement du programme (c'est-à-dire pendant l'enregistrement du programme) mais *pendant* le déroulement du programme.

Le problème posé peut être facilement résolu par deux utilisations successives de l'instruction INP destinées à permettre l'introduction, en cours de programme, de W_2 puis W_1 . Encore faut-il avoir le temps matériel de placer le nombre W_1 sur les clés D_7 à D_0 entre les deux instructions INP. Ceci n'est concevable que si la fréquence d'horloge est très basse où si l'on travaille en pas à pas. L'établissement du résultat requiert alors beaucoup de temps : cette solution n'est pas intéressante. Elle doit être obligatoirement abandonnée si la fréquence d'horloge est fixée à une valeur élevée (par exemple, utilisation d'un quartz 1 MHz).

Une autre solution est envisagée dans ce chapitre : elle consiste à *arrêter* quand il le faut le déroulement du programme ; l'expérimentateur a alors tout le temps pour présenter W_1 sur les clés D_7 à D_0 . C'est lui (ou un dispositif automatique) qui commande le redémarrage du séquençement lorsque la donnée à « entrer » est prête sur les clés. Il y a eu *interruption de programme, en vue du chargement direct d'une donnée dans la mémoire* ($\overline{\text{DMA-IN}}$) présentée par les clés D_7 à D_0 .

II. Mise en œuvre de l'interruption par le canal $\overline{\text{DMA-IN}}$

II.1. L'instruction IDLE

Nous avons utilisé plusieurs fois cette instruction pour arrêter le séquençement lorsqu'un programme est achevé.

Supposons que le code 00 de l'instruction IDLE soit inscrit à l'adresse Z de la mémoire. Le cycle de recherche de cette instruction conduit au chargement du code 00 dans les registres I et N du microprocesseur et à l'incréméntation du compteur de programme (registre R(P)) dont le contenu passe à $Z + 1$. Pendant le cycle d'exécution, la mémoire est placée en fonction lecture et M(R(0)) apparaît sur le bus des données (propriété que nous n'exploitons pas ici). Le cycle qui suit n'est *jamais* un cycle de recherche.

Supposons qu'il y ait, pendant le cycle d'exécution de IDLE, présentation d'une requête d'interruption. Pratiquement, cette requête est faite en plaçant $\overline{\text{DMA-IN}}$ ou $\overline{\text{DMA-OUT}}$ ou $\overline{\text{INTERRUPT}}$ à 0. Pour que cette requête soit acceptée, il faut qu'elle soit présente lorsque $\text{TPB} = 1$. Dans ce cas, à l'issue du cycle d'exécution de IDLE, le microprocesseur s'engage, suivant le cas, dans un cycle DMA ou INTERRUPT et le fait savoir en envoyant sur les lignes SC_1 et SC_0 soit le code (1,0) soit le code (1,1).

S'il n'y a pas, pendant le cycle d'exécution de IDLE, de présentation et acceptation de requête d'interruption, ce cycle d'exécution sera suivi d'autres cycles d'exécutions identiques. Le code d'état demeure donc $(\text{SC}_1, \text{SC}_0) = (0,1)$. Le microprocesseur est ainsi mis dans un état d'attente, d'où il ne sortira que lorsqu'une demande d'interruption sera présentée.

II.2. Conséquence de l'acceptation d'une requête de $\overline{\text{DMA-IN}}$

Nous nous intéressons ici uniquement au cas où une requête de DMA a été présentée, en vue d'entrer une donnée dans la mémoire. Pour cela $\overline{\text{DMA-IN}}$ a été mis à 0 lorsque $\text{TPB} = 1$. Dans le cycle DMA qui suit, la mémoire est adressée par le registre R(0) et est placée en fonction écriture ($\overline{\text{MWR}} = 0$) : le mot présent sur le bus des données (présenté par les clés D_7 à D_0 car le microprocesseur est en HI) est alors enregistré à l'adresse R(0). Puis le registre R(0) s'incréménte.

Un autre cycle DMA suivra si, pendant celui-ci, $\overline{\text{DMA-IN}}$ a été maintenu à 0 et conduira à l'enregistrement d'une donnée à l'adresse suivante de la mémoire. Ce fonctionnement a été mis à profit pour enregistrer un programme complet ; il ne nous intéresse plus ici. Faisons donc l'hypothèse que $\overline{\text{DMA-IN}}$ repasse à 1 dès que la requête a été acceptée, c'est-à-dire au début du premier cycle DMA-IN.

A l'issue du cycle DMA-IN, le microprocesseur reprend son fonctionnement normal (mode RUN). Le cycle suivant est un cycle de recherche : le code écrit à l'adresse fournie par le compteur de programme est transféré dans les registres I et N,

Il s'agit du code écrit à l'adresse $Z + 1$ puisque c'est avec ce contenu que nous avons laissé le compteur de programme à l'issue du cycle de recherche de IDLE. Après l'incréméntation du compteur de programme, l'instruction dont le code était inscrit à l'adresse $Z + 1$ s'exécute. Et ainsi de suite... La figure 1 résume la chronologie des opérations qui viennent d'être décrites.

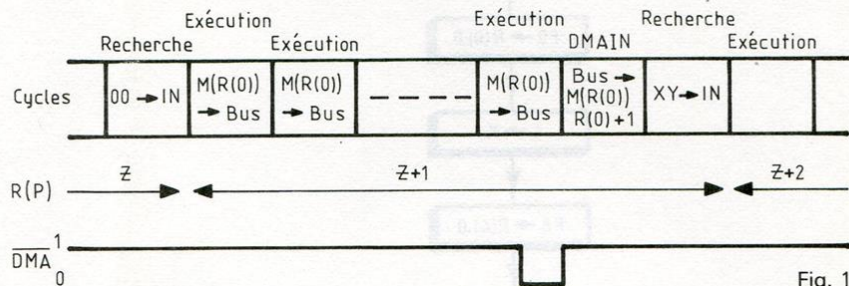


Fig. 1

III. Solution proposée au problème énoncé

III. 1. Solution provisoire

a) Etablissement d'un organigramme

Il faut avoir présentes à l'esprit les considérations suivantes :

- à l'issue d'une initialisation, on sait que le registre $R(0)$ est obligatoirement compteur de programme et que son contenu est 00. Or, on a besoin du registre $R(0)$ pour pointer la mémoire lors du fonctionnement DMA-IN en cours de programme. Il est donc nécessaire de changer de compteur de programme. Nous avons attribué ce rôle au registre $R(3)$. Il est nécessaire d'initialiser le contenu du registre $R(3)$ (voir chapitre 13).

- pour l'utilisation des instructions d'entrée et sortie, il est nécessaire de disposer d'un pointeur de données. Pourquoi ne pas prendre le registre $R(4)$? Nous l'initialisons à F8 : notre programme n'atteindra pas cette adresse.

- il est nécessaire de fixer l'adresse à laquelle va être stockée la donnée W_1 grâce au cycle DMA-IN que nous prévoyons. Le contenu du pointeur de DMA (registre $R(0)$) doit donc être fixé avant l'introduction de W_1 par exemple à FO.

- bien entendu, les initialisations ne portent que sur la partie de faible poids des registres $R(3)$, $R(4)$ et $R(0)$ puisque la mémoire utilisée ne comporte que 256 lignes.

- un cycle DMA-IN comporte une incréméntation du registre pointeur $R(0)$. Si l'on veut utiliser la donnée qui est stockée au cours de cette opération, il est nécessaire de compenser cette incréméntation, alors inutile et fâcheuse, par une décréméntation.

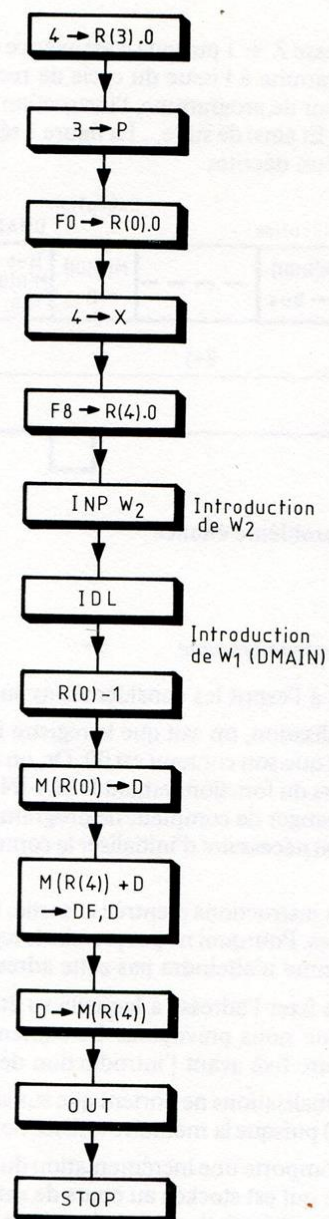


Fig. 2

Cette solution suppose que W_2 est présent sur les clés au moment du lancement du programme et est chargé sur le registre D et en mémoire grâce à l'instruction INP1 qui possède la propriété d'activer à 1 la ligne de sortie de contrôle N_0 . Le coupleur de sortie est donc activé et la donnée W_1 est en même temps ressortie et affichée.

Nous avons utilisé l'addition comme exemple d'opération. Par soucis de simplification, le contenu du registre DF n'a pas été visualisé sur la sortie Q.

b) écriture du programme (fig. 3).

Pour faciliter l'enregistrement du programme, le registre R(3).0 a été initialisé à 04. Les parties de programme où le registre R(0) est compteur et où le registre R(3) est compteur sont alors adjacentes.

L'instruction LOAD VIA N ne peut pas être utilisée pour $N = 0$. Nous l'avons remplacée par LOAD ADVANCE et compensé l'incréméntation du registre R(0), non souhaitée, par une décréméntation.

Adresse	Code et opérande	Mnémonique	Commentaire
00	F8 04	LDI	} 04 \rightarrow R(3).0
02	A3	PLO R(3)	
03	D3	SEP R(3)	
			Registre R(3) devient compteur de programme
04	F8 F0	LDI	} F0 \rightarrow R(0).0
06	A0	PLO R(0)	
07	E4	SEX R(4)	
			4 \rightarrow X : registre R(4) := pointeur de données
08	F8 F8	LDI	} F8 \rightarrow R(4).0
0A	A4	PLO R(4)	
0B	69	INP 1	
			entrée et affichage de W_2
0C	00	IDL	attente pour DMA IN
0D	20	DEC R(0)	R(0) - 1
0E	40	LDA	} M(R(0)) = $W_1 \rightarrow$ D
0F	20	DEC R(0)	
10	F4	ADD	$W_1 + W_2 \rightarrow$ D
11	54	STR	D = $W_1 + W_2 \rightarrow$ M(R(4)) = M(F8)
12	61	OUT 1	M(R(4)) = $W_1 + W_2 \rightarrow$ afficheurs
13	00	IDL	attente

Fig. 3

c) essai du programme

Essai pas à pas

Le programme étant enregistré, *il faut* enlever la liaison de \overline{SR} à $\overline{DMA-IN}$, placer le commutateur d'horloge en position « pas à pas » et le commutateur RAM ROM sur la position RAM, faire ensuite $\overline{WAIT} = 1$ puis $\overline{CLEAR} = 1$.

Faire dérouler le programme jusqu'à la recherche de l'instruction IDL. Le poids faible 0C de l'adresse où est enregistrée cette instruction apparaît sur les afficheurs d'adresses dans la deuxième partie de ce cycle pour lequel $(SC_1, SC_0) = (0,0)$. Dans le cycle d'exécution qui suit $((SC_1, SC_0) = (0,1))$, il apparaît sur les afficheurs d'adresses le contenu du registre R (0) soit 00F0 : en effet ce registre est utilisé pour pointer la mémoire, alors en fonction lecture, de sorte que $M(R(0)) \rightarrow$ bus. Vérifier que ce cycle d'exécution se reproduit tant que l'on n'intervient pas.

Présenter W_1 sur les clés de programmation

A n'importe quel instant d'un quelconque cycle d'exécution de IDL, faire et maintenir $\overline{DMA-IN} = 0$. Fournir des impulsions d'horloge jusqu'à ce que le code d'état passe à $(SC_1, SC_0) = (1,0)$ indiquant ainsi que la requête de DMA est acceptée. Il faut alors revenir à $\overline{DMA-IN} = 1$, avant que TPB ne passe à 1 si l'on ne souhaite pas avoir un autre cycle DMA-IN. Pratiquement, faire $\overline{DMA-IN} = 1$ dès que (SC_1, SC_0) passe à (1,0). Remarquer que pendant l'exécution du cycle DMA-IN, la mémoire se place en fonction écriture et que le contenu du registre R (0), soit ici 00F0, apparaît en deux temps sur les afficheurs d'adresse : c'est bien à cette adresse que va se stocker en mémoire le nombre W_1 présenté par les clés sur le bus des données.

À l'issue de ce cycle, (SC_1, SC_0) passe à 00 et commence le cycle de recherche de l'instruction dont le code est inscrit à l'adresse 0D lisible sur les afficheurs d'adresses.

On peut terminer le déroulement du programme jusqu'à ce que le résultat $W_1 + W_2$ apparaisse sur les afficheurs de sortie.

Essai en mode astable

Laisser la liaison \overline{SR} à $\overline{DMA-IN}$: elle va être utilisée.

Faire $\overline{WAIT} = 1$. Présenter W_2 sur les clés de programmation. Faire $\overline{CLEAR} = 1$. Le programme se déroule jusqu'à l'instruction IDL dont l'exécution se répète tant que l'on n'intervient pas : SC_0 reste à 1.

On a tout son temps pour positionner les clés à W_1 . Cette opération terminée, il suffit d'appuyer sur le bouton poussoir : la sortie \overline{SR} de la bascule JK passe à 0, donc $\overline{DMA-IN}$ passe à 0, ce qui constitue la requête de DMA. Lorsque cette requête est acceptée, SC_1 passe à 1 et entraîne la remise à 1 de $\overline{DMA-IN}$ (revoir à ce sujet le chapitre 7 de la 2e partie). Un seul cycle DMA-IN se déroule, suivi de la succession normale recherche - exécution - recherche... jusqu'à la fin du programme.

III. 2. Solution définitive

La solution précédente utilise deux registres pointeurs : R (0) et R (4). Pourquoi ne pas utiliser un seul registre pointeur, qui sera forcément R (0) puisque le mode DMA l'impose ? On est donc amené à fixer X à 0.

Pourquoi ne pas entrer W_2 par la même procédure que W_1 , c'est-à-dire en interrompant le déroulement du programme afin de faire un DMA-IN ? Dans ce cas, pour éviter d'écrire deux fois des instructions identiques, on peut penser faire une boucle. Cette solution nous amène tout naturellement à penser à une extension des possibilités du système : on peut essayer de réaliser un dispositif permettant d'additionner autant d'octets que l'on désire, les résultats intermédiaires étant lisibles sur les afficheurs de sortie.

Pendant que nous y sommes, prévoyons la visualisation d'une retenue éventuelle. La solution a déjà été exposée.

Toutes ces considérations nous conduisent à l'organigramme de la figure 4 et le programme correspondant de la figure 5, où l'on remarquera que :

- le registre D est mis à 0 en début de programme. Dans la première description du programme entre deux instructions IDL, le premier nombre présenté sur les clés est ainsi additionné à 0 avant d'être affiché.

- nous n'avons pas initialisé le registre R (0) mais nous avons pris soin de laisser libre un espace de la mémoire situé après le code de SEP R (3). Après exécution de cette fonction, le registre R (0) est un effet abandonné avec le contenu $0003 + 1 = 0004$. Le chargement de la première donnée par le biais du DMA-IN se fait donc à l'adresse 04. Dans la suite du programme, ce registre est géré de manière qu'il pointe toujours, lorsque l'on en a besoin, la ligne d'adresse 04 de la mémoire (il est décrémenté à deux reprises pour compenser deux incrémentations non souhaitées lors d'un cycle DMA-IN et de l'exécution de OUT).

Si vous en avez la possibilité, enregistrez ce programme. Essayez-le en faisant des additions successives, comme à l'aide d'une calculatrice banale.

Attention, la capacité maximale est 1 FF. Vous pouvez imaginer des programmes permettant de traiter des nombres plus grands en faisant un affichage du résultat en plusieurs fois.

IV. Autre exemple d'interruption par le canal DMA-IN

IV. 1. Énoncé d'un problème

Un moteur doit tourner pendant un nombre de tours W programmable à l'avance. Le nombre W (compris entre 1 et 255) est présenté par les clés D_7 à D_0 . Une came, solidaire de l'axe de rotation, ferme un contact à chaque tour, plaçant ainsi \overline{EF}_1 à 0 (voir fig. 6). Par ailleurs,

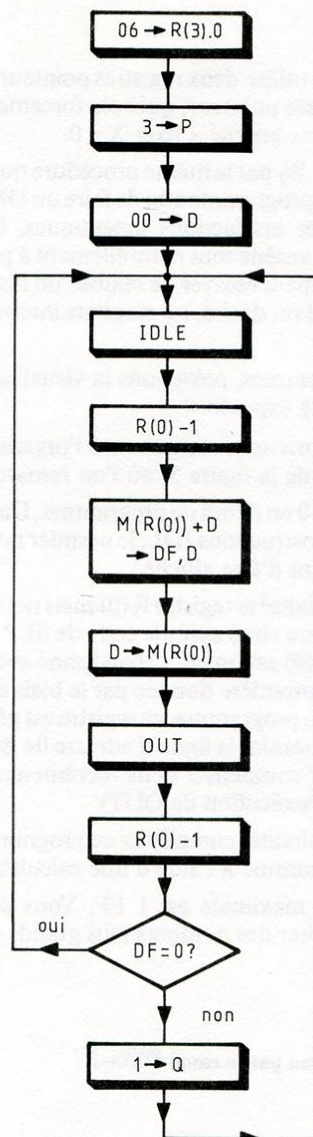


Fig. 4

Adresse	Code et op�r�nde	Mn�monique	Commentaire
00	F8 06	LDI	} 06 \rightarrow R(3).O Reg. R(3) = compteur de p.
02	A3	PLO R(3)	
03	D3	SEP R(3)	
04	��		} adresses libres pour recevoir donn�es
05	��		
06	E0	SEX R(0)	O \rightarrow X : R(0) = pointeur
07	F8 00	LDI	00 \rightarrow D
09	00	IDL	attente pour DMA IN
0A	20	DEC R(0)	R(0) - 1 \Rightarrow R(0).O = 04
0B	F4	ADD	M(R(0)) + D \rightarrow DF, D
0C	50	STR	D \rightarrow M(R(0)) = M(04)
0D	61	OUT	M(04) \rightarrow Afficheurs
0E	20	DEC R(0)	R(0) - 1 \Rightarrow R(0).O = 04
0F	3B 09	BNF	Branch. � IDL si D = 0
11	7B	SEQ	1 \rightarrow Q
12	30 09	BR	Branch. � IDL

Fig. 5

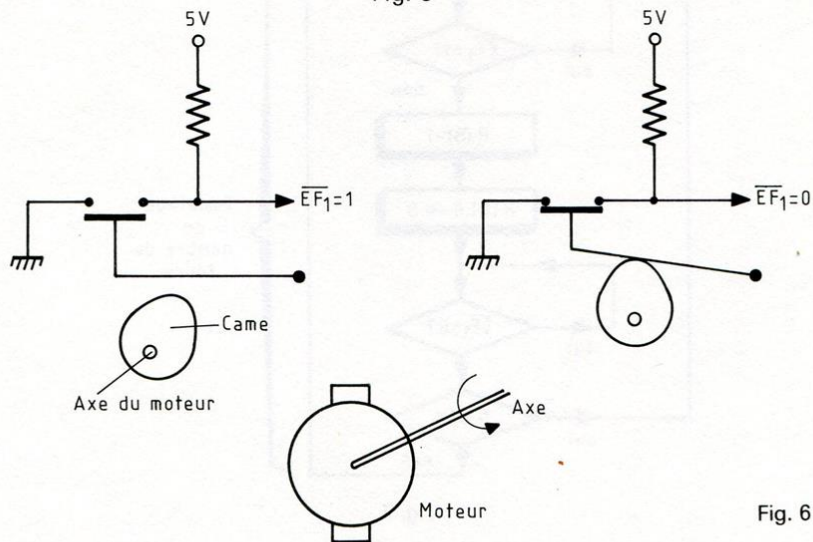


Fig. 6

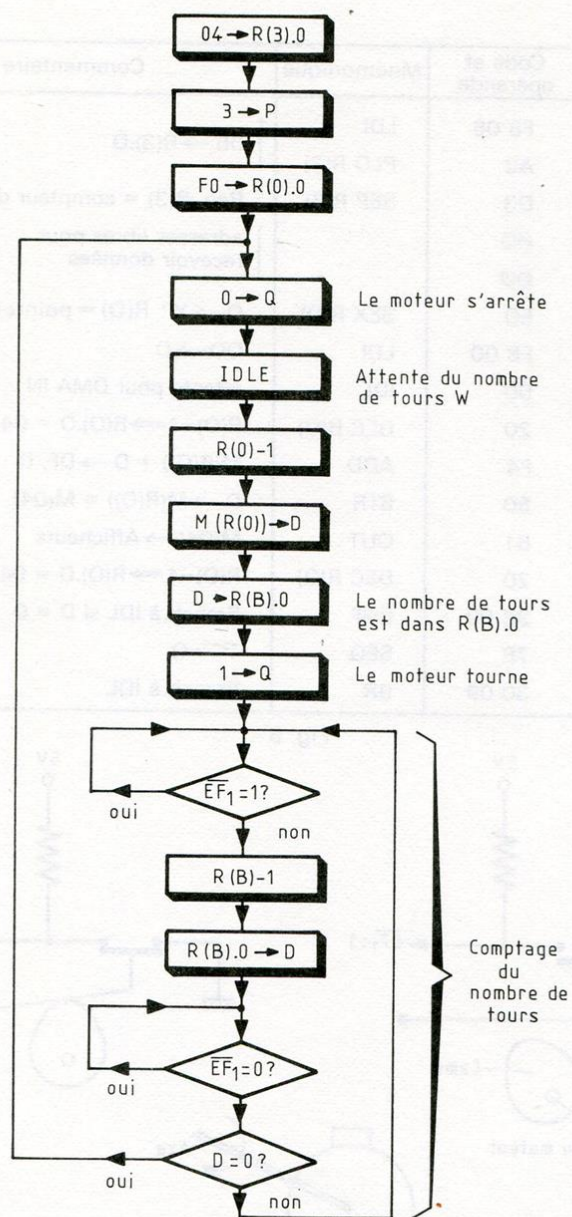


Fig. 7

Adresse	Code et opérande	Mnémonique	Commentaire
00	F8 04	LDI	} initialisation du futur compteur de programme 04 → R(3).0
02	A3	PLO R(3)	
03	D3	SEP R(3)	
04	F8 F0	LDI	} initialisation du pointeur en DMA F0 → R(0).0
06	A0	PLO R(0)	
07	7A	REQ	
08	00	IDL	O → Q : le moteur s'arrête attente de DMA IN (du nombre de tours W)
09	20	DEC R(0)	} W = M(R(0)) → D
0A	40	LDA R(0)	
0B	20	DEC R(0)	
0C	AB	PLO R(B)	W = D → R(B).0
0D	7Q	SEP	1 → Q : le moteur tourne
0E	3C 0E	BN1	programme bloqué tant que $\overline{EF}_1 = 1$
10	2B	DEC R(B)	
11	8B	GLO R(B)	
12	34 12	B1	programme bloqué tant que $\overline{EF}_1 = 0$
14	32 07	BZ	branchement à 07 si D = 0
16	30 0E	BR	branchement à 0E

Fig. 8

le moteur ne doit pas tourner si le nombre de tours n'est pas programmé. On suppose que lorsque $Q = 0$ le moteur est à l'arrêt et la mise à 1 de Q entraîne sa mise en rotation.

Le fréquence de l'horloge du microprocesseur est élevée mais la vitesse de rotation du moteur est faible : il faut prévoir qu'il s'écoule de nombreuses périodes d'horloge pendant que \overline{EF}_1 est maintenu à 0 à chaque tour.

IV. 2. Solution proposée (voir organigramme et programme des figures 7 et 8).

Le nombre de tours est programmé alors que le moteur est à l'arrêt ($Q = 0$) et le microprocesseur en état d'attente (IDLE) d'où l'on sort en faisant $\overline{DMA-IN} = 0$. Une décrémentation du registre R(0) compense celle qui est intervenue dans le cycle de DMA-IN. Le nombre W, chargé dans la mémoire à l'adresse R(0) pendant le cycle de DMA-IN est transféré sur le registre R(B).0 par l'intermédiaire du registre D. Le moteur est alors mis en route ($Q = 1$).

Lorsque \overline{EF}_1 passe à 0, le registre R (B) est décrémenté et son état est testé (par l'intermédiaire du registre D), après que \overline{EF}_1 soit revenu à 1. Si $R(B).0 = D = 0$, le moteur s'arrête ($Q = 0$) et le microprocesseur est placé dans l'état d'attente. Si $R(B).0 = D \neq 0$, le moteur continue de tourner et une nouvelle décrémentation de R (B) se produit dès que \overline{EF}_1 reprend la valeur 0.

Bien entendu, l'utilisation en cours de programme de DMA-IN nous oblige à utiliser un autre registre que le registre R (0) comme compteur de programme. Le début de programme est consacré aux opérations d'initialisation du compteur de programme (registre R (3)), au changement du compteur et à l'initialisation du pointeur dans le mode DMA (registre R (0)).

V. Note complémentaire

Nous avons utilisé l'instruction IDLE avant de procéder à une requête de DAM-IN afin seulement d'avoir le temps matériel de présenter sur les clés la donnée à enregistrer en mémoire (nombre à additionner ou nombre de tours).

En fait, une requête de DMA-IN peut être présentée à tout moment *au cours* du déroulement d'un programme. Le cycle en cours (de recherche ou d'exécution) est achevé ; le cycle qui suit est alors obligatoirement un cycle DMA-IN pendant lequel la donnée présente sur le bus est stockée en mémoire à l'adresse indiquée par le registre R (0). Si pendant ce cycle DMA-IN une nouvelle requête n'a pas été présentée, le déroulement du programme reprend normalement.

Exemple : réalisation d'une bascule astable de période réglable (rapport cyclique 1/2).

Nous avons déjà élaboré une solution de ce problème en utilisant l'instruction INPUT (voir chapitre 14). Une solution empruntant le canal $\overline{DMA-IN}$ est décrite par l'organigramme de la figure 9.

Le registre R (3) a été choisi pour jouer le rôle de compteur de programme et le registre R (B) reçoit le nombre W à décrémenter pour réaliser une temporisation θ . Ce nombre W, présenté sur les clés D_7 à D_0 , est chargé dans la mémoire à l'adresse R (0), *à n'importe quel moment du déroulement du programme*, par simple pression sur le bouton poussoir. (Ceci suppose bien entendu que l'on travaille dans le mode « astable » et que l'on a réalisé la liaison $\overline{SR} - \overline{DMA-IN}$). On remarque qu'il est nécessaire d'attendre la fin de la demi-période en cours pour que la modification de la période puisse se faire. Au lancement du programme, et tant qu'il n'a pas été provoqué un cycle DMA-IN afin de charger un nombre W déterminé, la période d'oscillation est aléatoire : elle dépend du nombre qui se trouve à l'adresse correspondant au contenu du registre R (0), ce nombre étant obligatoirement inconnu.

Un autre problème se pose : que se passe-t-il si l'on provoque plusieurs cycles DMA-IN au cours d'une même demi période de fonctionnement ? Supposons que $R(0).0$ ait été initialisé à XY. Un premier $\overline{DMA-IN} = 0$ charge W_1 à l'adresse XY et incrémente le registre pointeur R (0). Un deuxième $\overline{DMA-IN} = 0$ charge W_2 à

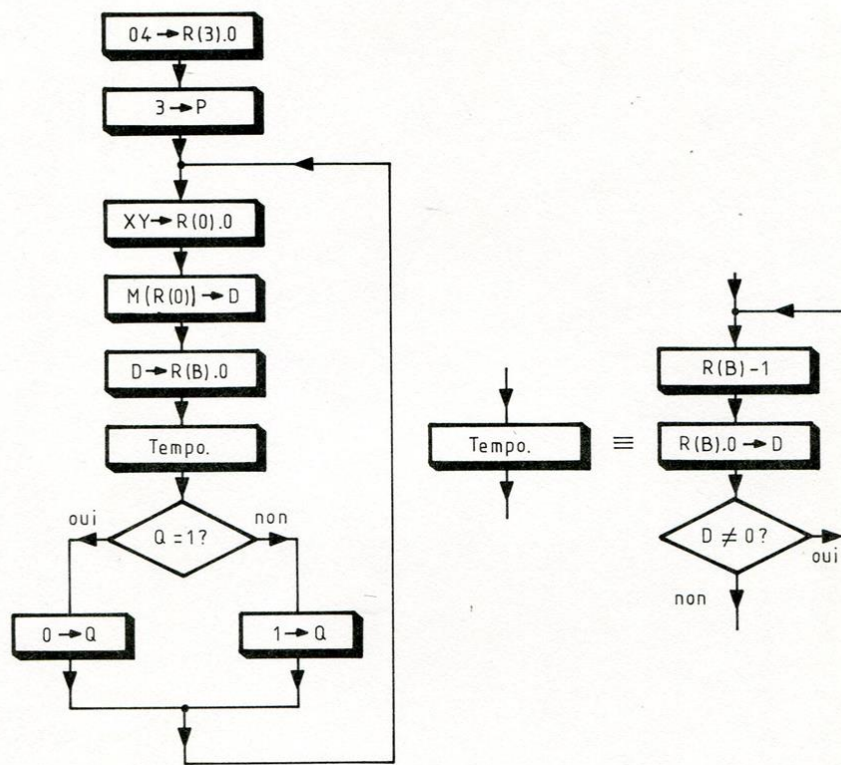


Fig. 9

l'adresse $XY + 1$ et incrémente à nouveau le registre $R(0)$ de sorte que un troisième $\overline{DMA-IN} = 0$ charge W_3 à l'adresse $XY + 2$, et ainsi de suite éventuellement. Tel que nous avons prévu l'organigramme, le registre $R(0).0$ est réinitialisé à XY avant le chargement du registre D . C'est donc le nombre W_1 qui va être chargé dans le compteur de boucle $R(B).0$ et c'est de ce nombre W_1 que va dépendre la future demi-période. Les nombres W_2, W_3, \dots sont ignorés ce qui peut constituer un défaut important du système.

Nous laissons au lecteur le soin d'établir le programme correspondant et, éventuellement, de l'essayer afin de mettre en évidence les qualités et les défauts de ce système. Il peut également prévoir l'affichage du nombre W dont la période dépend.

Chapitre 16

L'INTERRUPTION PAR LE CANAL INTERRUPT

I. Position d'un problème et éléments de résolution

Nous voulons à nouveau réaliser une bascule astable de rapport cyclique 1/2 et de fréquence réglable. Les exemples traités dans les deux chapitres précédents conduisent à une prise en compte de la modification de fréquence demandée qui n'est pas immédiate. Dans le cas de l'utilisation de l'instruction INPUT, la modification souhaitée n'apparaît qu'au moment où l'instruction INPUT est exécutée. Dans le cas de l'utilisation du mode DMA-IN, si la nouvelle période (au travers d'un nouveau nombre W à décrémenter) est bien immédiatement enregistrée grâce à une interruption du déroulement du programme laissant s'intercaler un cycle DMA IN, sa prise en compte effective n'intervient que lors du transfert $M(R(0)) \rightarrow D$. Dans les deux cas, il faut attendre que le programme temporisateur en cours se termine avant que le nouveau nombre W soit chargé dans le compteur de boucle.

Nous nous imposons maintenant de pouvoir modifier *immédiatement* la période d'oscillation, dès qu'une requête est présentée, sans avoir à attendre que la temporisation en cours soit terminée.

La solution d'un tel problème ne peut être trouvée que s'il existe une possibilité d'interrompre à tout moment le déroulement du programme permettant d'introduire le nombre W caractérisant la nouvelle période (l'interruption par le canal DMA IN suffit pour satisfaire à cette exigence) *et* de réaliser des fonctions supplémentaires. Précisément, le microprocesseur COSMAC est conçu pour qu'il soit possible d'interrompre à tout moment le déroulement d'un programme pour permettre au microprocesseur d'effectuer des opérations dont la succession est programmée dans un *sous-programme d'interruption* indépendant. Lorsque ce sous-programme est traité, l'interruption se termine par un retour au programme principal qui continue alors de se dérouler en tenant compte s'il y a lieu, des éléments apportés lors de l'interruption.

II. Mise en œuvre de l'interruption par le canal INTERRUPT

I. 1. Présentation et acceptation d'une requête d'interruption par la canal INTERRUPT

Une requête d'interruption peut être présentée à n'importe quel moment du déroulement du programme en plaçant l'entrée de contrôle INTERRUPT au niveau logique 0.

Pour que cette requête d'interruption par le canal INTERRUPT soit acceptée, il faut que :

- le niveau logique 0 soit maintenu sur l'entrée INTERRUPT jusqu'au moment de l'apparition de $TPB = 1$ du (ou du dernier) *cycle d'exécution* de l'instruction en cours de réalisation : cette interruption n'est éventuellement prise en compte qu'à la fin de la réalisation d'une instruction.

- la bascule IE (interrupt enable ou autorisation d'interruption) soit à l'état 1.

Le cas échéant, à l'issue du (ou du deuxième) cycle d'exécution de l'instruction en cours, le microprocesseur s'engage dans un cycle particulier que nous appellerons cycle d'interruption, et le fait savoir en envoyant le code (1,1) sur les lignes (SC_1 , SC_0).

Notons que le *compteur de programme* contient alors l'adresse Z de l'instruction qui serait recherchée si l'interruption n'était pas servie.

N.B. Si un cycle DMA IN suit indifféremment un cycle de recherche ou un cycle d'exécution (ou même éventuellement un cycle d'interruption) un cycle d'interruption suit toujours un cycle d'exécution et précède un cycle de recherche (ou bien quelquefois un cycle DMA.)

I. 2. Conséquences de l'acceptation d'une requête d'interruption par le canal INTERRUPT

Dans le cycle d'interruption suivant l'acceptation d'une requête, la mémoire est placée dans l'état H.I. Aucune opération de lecture ou d'écriture ne se fait. Les opérations effectuées, internes au microprocesseur sont :

- les contenus des registres X et P (c'est-à-dire les numéros des registres pointeur de données et compteur de programme du programme interrompu) sont stockés dans le registre temporaire T. Soit $(X,P) \rightarrow T$.

- le contenu du registre X est fixé à 2 : le registre R(2) devient donc pointeur de données. Soit $2 \rightarrow X$.

- le contenu du registre P est fixé à 1 : le registre R(1) devient donc compteur du programme. Soit $1 \rightarrow P$.

– la bascule interne IE est mise à 0 ce qui interdit la prise en considération d'une nouvelle demande d'interruption par le canal INTERRUPT. Notez cependant qu'une demande de DMA IN est toujours possible.

En définitive, ce cycle d'interruption constitue un appel à un sous-programme (sous programme d'interruption) dont le compteur est obligatoirement le registre R (1) et dont le pointeur de données est le registre R (2). Les numéros des registres caractéristiques du programme interrompu (registre pointeur de données et registre compteur de programme) ne sont pas perdus puisqu'ils sont placés dans le registre T, laissant ainsi entrevoir une possibilité de retour à ce programme interrompu.

1. 3. Déroulement du sous-programme d'interruption et retour au programme interrompu

Le cycle suivant le cycle d'interruption est généralement un cycle de recherche (exceptionnellement un cycle DMA, jamais un cycle d'exécution). Le compteur de programme étant maintenant le registre R (1), c'est le code de l'instruction écrit à l'adresse pointée par ce registre R (1) qui est transféré sur les registres I et N du microprocesseur et c'est l'instruction correspondante qui est exécutée pendant le cycle suivant. Ainsi débute le déroulement du sous-programme d'interruption.

Lorsque le sous-programme d'interruption a rempli sa tâche, il faut revenir au programme interrompu. Ceci pourrait être réalisé en restituant simplement aux registres X et P leur contenu initial stocké dans le registre T en faisant l'opération $T \rightarrow (X, P)$. Cette opération ne se fait pas directement, elle se fait par l'intermédiaire de la mémoire au moyen de deux instructions :

– SAVE (SAV, code 78) qui permet le stockage du contenu du registre T (c'est-à-dire ici les numéros des registres R respectivement pointeur de données et compteur de programme du programme interrompu) dans la mémoire à la ligne pointée par le registre R (X) (ici registre R (2)). Soit $T \rightarrow M(R(X))$.

– RETURN (RET, code 70) qui produit le chargement du mot $M(R(X))$ dans les registres X et P. Ainsi les registres X et P retrouvent leur contenu initial. En plus, l'exécution de cette instruction remet la bascule IE à l'état 1 (autorisant ainsi la prise en considération d'une requête d'interruption ultérieure) et conduit à l'incréméntation du registre R(X) (ce qui n'est pas toujours souhaité). En résumé RET produit $M(R(X)) \rightarrow (X, P), R(X) + 1, 1 \rightarrow IE$.

Remarques

1. Le contenu du compteur de programme du programme interrompu est demeuré Z : à l'issue de cette interruption, le programme reprend donc par la recherche de l'instruction dont le code est à l'adresse Z de la mémoire.

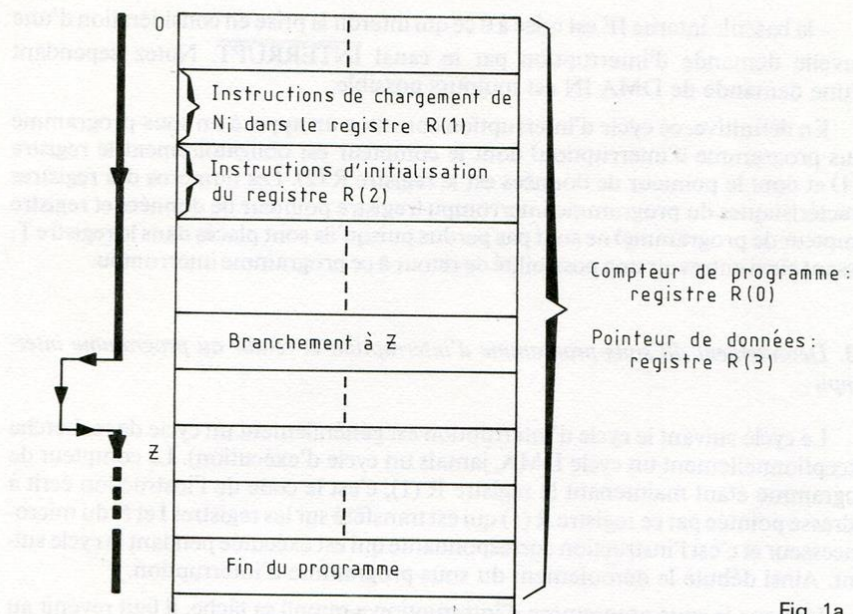


Fig. 1a

2. Bien entendu, le programme principal doit comporter une initialisation des registres R (1) (à l'adresse du début du sous-programme d'interruption) et R (2) (à l'adresse du bas d'une zone de la mémoire RAM, libre pour recevoir des données).

3. Si l'on ne souhaite pas qu'une nouvelle demande d'interruption ultérieure puisse être acceptée, il suffit de remplacer l'instruction RETURN par l'instruction DISABLE (DIS, code 71) substituant à l'opération $1 \rightarrow IE$, l'opération $0 \rightarrow IE$.

4. Si l'instruction RET est utilisée, c'est qu'une autre demande d'interruption par le canal $\overline{INTERRUPT}$ peut intervenir et doit être acceptée. Il faut alors que le compteur de sous-programme d'interruption soit réinitialisé c'est-à-dire qu'il contienne, au moment où la requête est acceptée, l'adresse N_i du début du sous-programme. Ceci est automatiquement réalisé si l'on s'arrange pour placer le code de l'instruction RET à la ligne d'adresse $N_i - 1$. Lors du cycle de recherche de RET, le compteur de sous programme d'interruption est incrémenté et son contenu passe à N_i . Le retour au programme interrompu le laisse dans cet état.

5. L'incrémentation du registre R (2) produite par l'instruction RET doit être compensée par une décrémentation. Celle ci peut être placée dès le début du sous-programme d'interruption.

A titre d'exemple, la figure 1 a indique le déroulement d'un programme comportant une instruction de branchement. La figure 1 b reprend le déroulement de ce

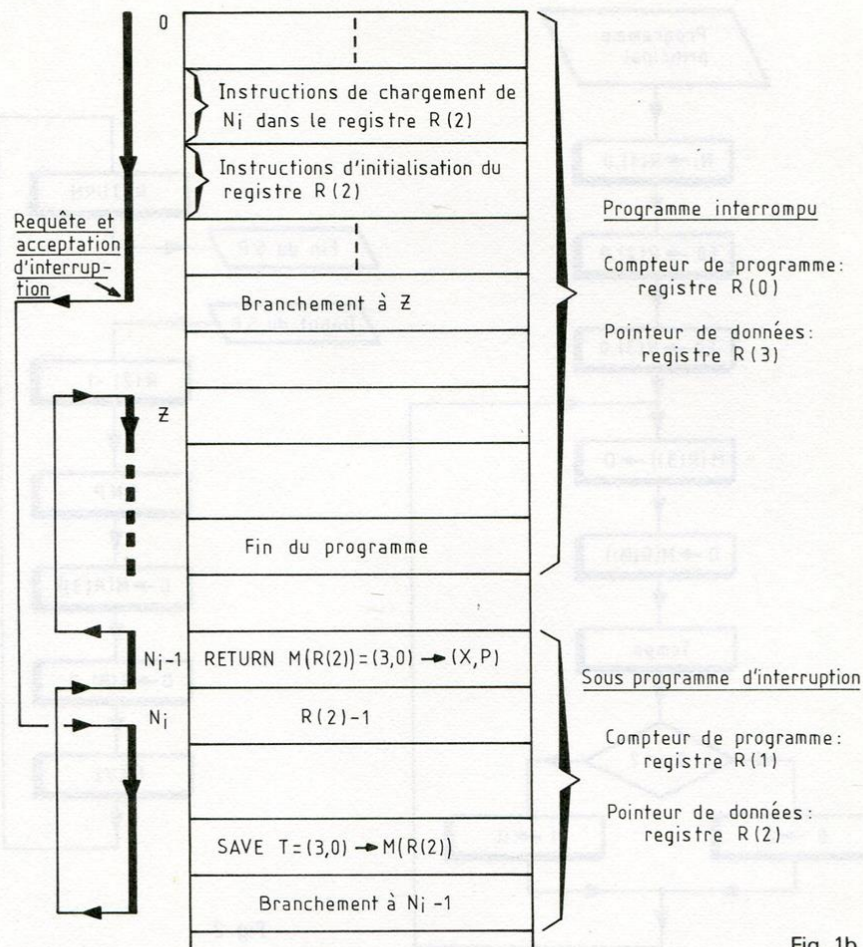


Fig. 1b

programme dans l'hypothèse où une demande d'interruption est faite et acceptée lors de la réalisation de l'instruction de branchement.

Elle met en évidence les relations existant entre le programme interrompu et le sous-programme d'interruption. Il faut bien noter que, à l'issue du cycle d'exécution de l'instruction de branchement, le compteur de programme (registre R(0)) contient Z mais le cycle suivant est le cycle d'interruption qui amène le changement de compteur de programme. L'instruction dont le code est inscrit à l'adresse Z n'est recherchée et exécutée qu'à l'issue du sous-programme d'interruption.

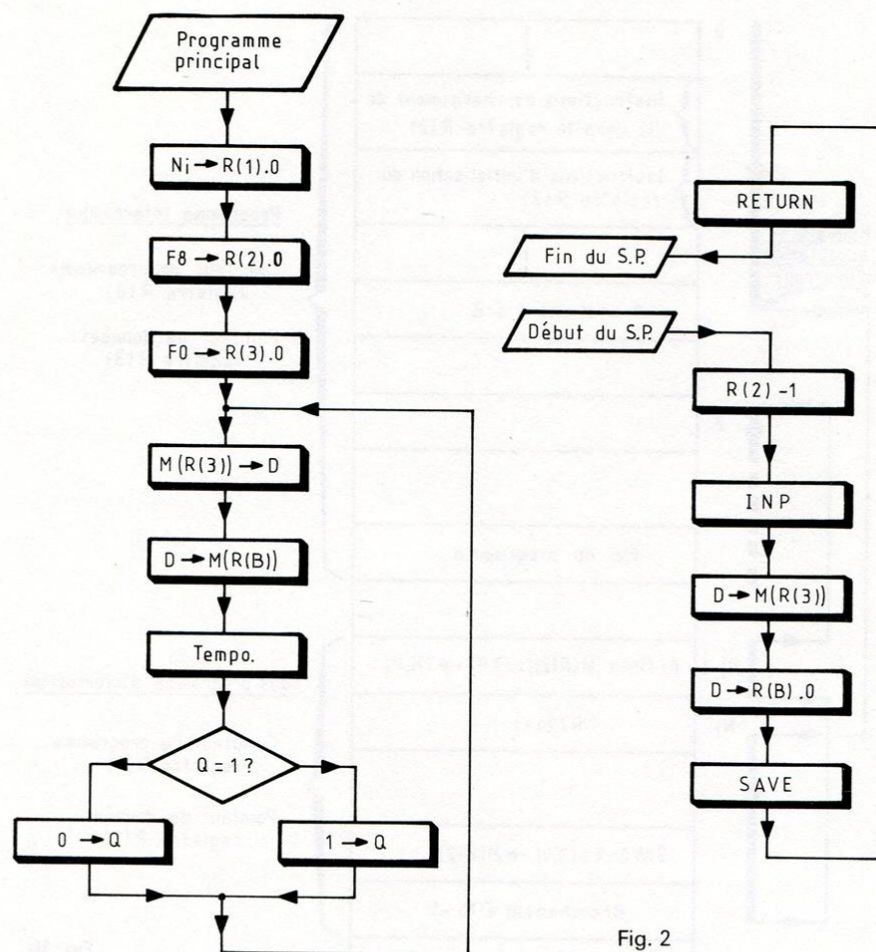


Fig. 2

III. Solution proposée au problème énoncé

III. 1. Etablissement d'un organigramme (fig. 2)

Comme on ne désire pas utiliser le mode DMA, le registre $R(0)$ est conservé comme compteur de programme. Le registre $R(3)$ est choisi comme pointeur de données dans ce programme. Il faut l'initialiser dès le début du programme.

L'interruption par le canal INTERRUPT va être utilisée. Il est donc nécessaire d'initialiser le compteur de programme (registre R (1)) et le pointeur de données (registre R(2)) du sous-programme d'interruption.

Le nombre W caractérisant la durée de temporisation, et donc la période d'oscillation, est lu à l'adresse R (3) de la mémoire. Au lancement du programme, ce nombre W est aléatoire. On a choisi le registre R (B) pour recevoir ce nombre et servir de compteur de boucle.

Le rôle du sous-programme d'interruption, auquel il peut être fait appel à tout moment, est double :

- il permet de charger immédiatement le compteur de boucle par le nouveau nombre W caractérisant la nouvelle période à fixer. Ainsi, même si une temporisation est en cours, elle est stoppée et reprend à l'issue de l'interruption sur la nouvelle base W.

- il conduit au stockage du nouveau nombre W à l'adresse R (3) de la mémoire pour assurer une durée convenable aux états haut et bas ultérieurs.

Notez la structure de ce sous-programme d'interruption, comportant obligatoirement les instructions DEC R (2), SAV et RET (ou DIS).

III. 2. Ecriture du programme

La figure 3 indique la façon dont on utilise l'espace disponible de la mémoire. Les zones réservées au programme principal et au sous-programme d'interruption sont adjacentes : ce n'est pas une obligation mais cela facilite la programmation.

Vous remarquerez que dans le programme correspondant de la figure 4, nous avons utilisé l'instruction INP 1. Ainsi le nombre W qui est entré pendant le sous-programme est en même temps affiché grâce à la validation du coupleur de sortie par $N_0 = 1$

III. 3. Essai éventuel du programme

a) En mode astable :

Lorsque le programme est enregistré, faire $\overline{\text{WAIT}} = 1$. Etablir la liaison entre $\overline{\text{SR}}$ et $\overline{\text{INTERRUPT}}$ par un cordon. Passer au mode RUN en faisant $\overline{\text{CLEAR}} = 1$: le programme se déroule et la bascule astable fonctionne à une fréquence aléatoire.

Pour fixer la période d'oscillation, présenter une valeur convenable de W sur les clés D_7 à D_0 . Appuyer à n'importe quel moment sur le bouton poussoir. L'entrée $\overline{\text{INTERRUPT}}$ est alors placée à 0 jusqu'à ce que le microprocesseur accepte la demande d'interruption (fin de la réalisation de l'instruction en cours). SC_1 passant

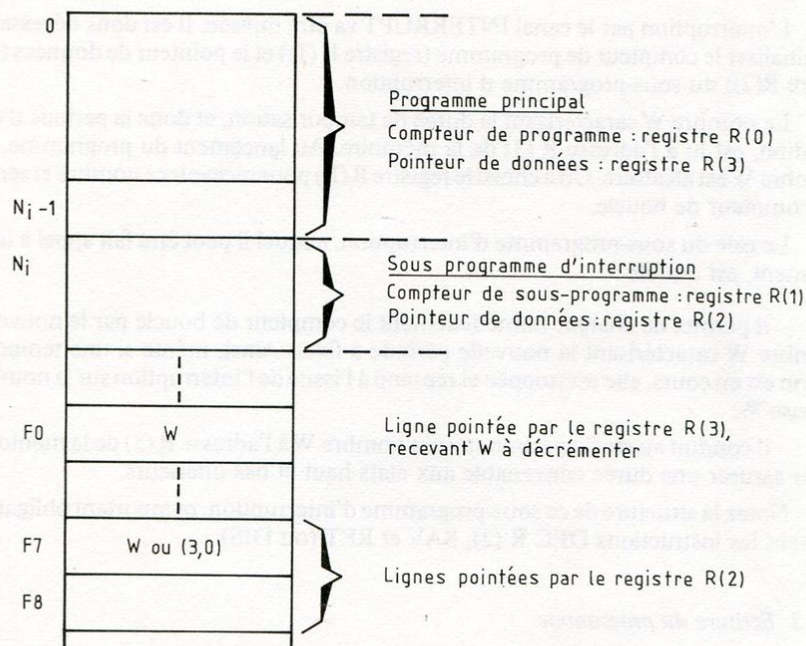


Fig. 3

alors à 1 remet automatiquement à 1 la sortie \overline{SR} . Le cycle d'interruption puis le sous-programme d'interruption se déroulent. Le nombre W présenté sur les clés s'affiche et la bascule oscille à la période correspondante.

b) En mode pas à pas

L'horloge est en mode pas à pas et \overline{SR} n'est pas reliée.

Commencer le déroulement du programme, présenter un nombre W faible sur les clés (2 par exemple) et faire une demande d'interruption en plaçant, grâce au commutateur correspondant, $\overline{INTERRUPT}$ à 0. Faire plusieurs essais afin de bien montrer que cette demande n'est acceptée que si $\overline{INTERRUPT}$ est à 0 lorsque TPB est à 1, uniquement dans le cycle d'exécution.

Observer que pendant le cycle d'interruption ($SC_1 = 1$ et $SC_0 = 1$), la mémoire reste en HI. Suivre bien toutes les opérations lors du déroulement du sous-programme d'interruption. Pendant le déroulement de ce sous-programme, présenter une

Adresse	Code et opérande	Mnémonique	Commentaire
00	F8 18	LDI	$N_i = 18 \rightarrow R(1).0$: initialisation du compteur de S.P d'interruption
02	A1	PLO R(1)	
03	F8 F8	LDI	
05	A2	PLO R(2)	$F8 \rightarrow R(2).0$: initialisation du pointeur de données du S.P.d'interruption
06	F8 F0	LDI	
08	A3	PLO R(3)	$F0 \rightarrow R(3).0$: initialisation du pointeur de données du P.P.
09	03	LDN	
0A	AB	PLO R(B)	$M(R(3)) = W \rightarrow D$ } chargement du $D \rightarrow R(B).0$ } compteur de boucles
0B	2B	DEC R(B)	
0C	8B	GLO R(B)	$R(B).0 \rightarrow D$
0D	3A 0B	BNZ	Branch. à 0B si $D \neq 0$
0F	31 14	BQ	Branch. à 14 si $Q = 1$
11	7B	SEQ	$1 \rightarrow Q$
12	30 15	BR	Branch. à 15
14	7A	REQ	
15	30 09	BR	Branch. à 09
17	70	RET	$M(R(2)) = (3,0) \rightarrow (X,P), R(2) + 1, 1 \rightarrow IE$: retour au p.p
→ 18	22	DEC R(2)	$R(2) - 1$; entrée du S.P d'interruption
19	69	INP 1	$W \rightarrow D$ et $W \rightarrow M(R(2)) = M(F7)$
1A	53	STR	$D = W \rightarrow M(R(3))$
1B	AB	PLO R(B)	$D = W \rightarrow R(B).0$
1C	78	SAV	$T = (3,0) \rightarrow M(R(2)) = M(F7)$
1D	30 17	BR	Branch. à 17 pour retour au P.P

Fig. 4

demande d'interruption en maintenant $\overline{INTERRUPT}$ à 0 ; elle n'est pas acceptée car $IE = 0$. Avant de revenir au programme interrompu, remettre $\overline{INTERRUPT}$ à 1. Constaté que le programme interrompu suit bien les indications enregistrées pendant le déroulement du sous-programme en décrivant deux ou trois cycles.

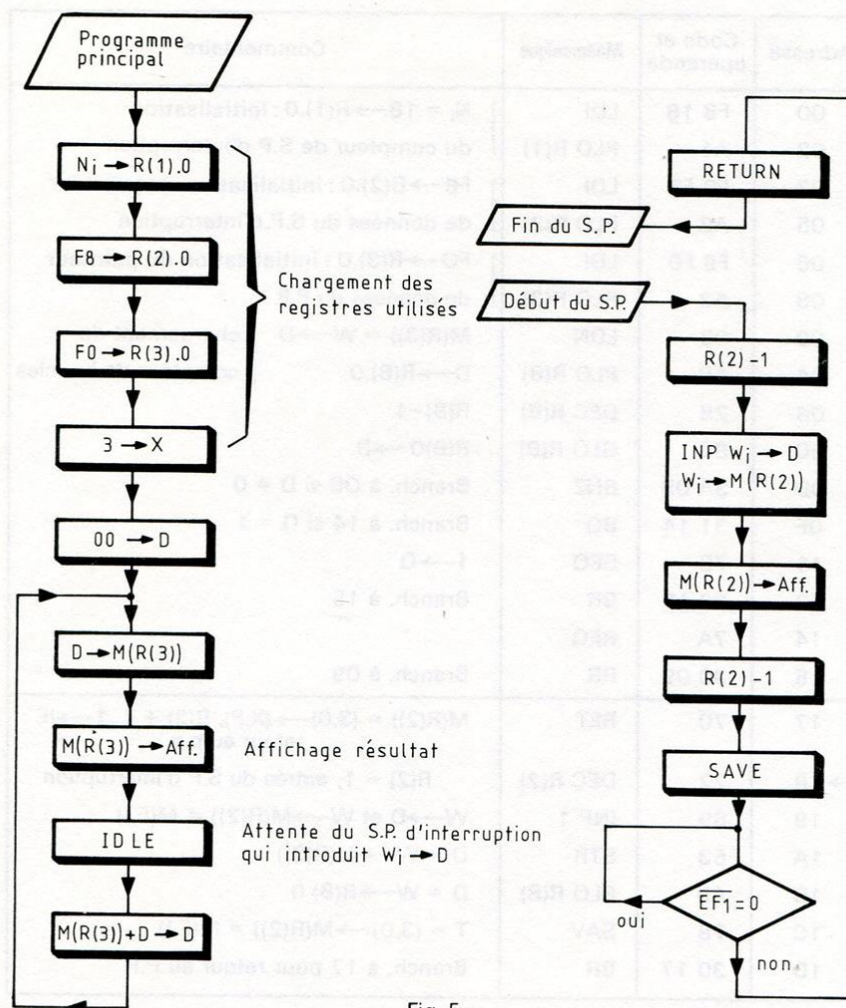


Fig. 5

IV. Autre exemple d'interruption par le canal INTERRUPT

IV. 1. Énoncé d'un problème

On veut réaliser un calculateur permettant d'effectuer les additions successives des nombres W_1, W_2, \dots, W_n avec les contraintes suivantes :

Adresse	Code et opérande	Mnémonique	Commentaire
00	F8 14	LDI	$N_i = 14 \rightarrow R(1).0$ initialisation du compteur de SP d'interruption
02	A1	PLO R(1)	
03	F8 F8	LDI	
05	A2	PLO R(2)	$F8 \rightarrow R(2).0$ initialisation du pointeur de données du SP d'interruption
06	F8 F0	LDI	
08	A3	PLO R(3)	$F0 \rightarrow R(3).0$ initialisation du pointeur de données du P.P.
09	E3	SEX R(3)	
0A	F8 00	LDI	$3 \rightarrow X$ désignation du pointeur de données du P.P. $00 \rightarrow D$
0C	53	STR	$D \rightarrow M(R(3))$
0D	61	OUT 1	Affichage et $R(3) + 1$
0E	23	DEC R(3)	$R(3) - 1$
0F	00	IDL	Attente d'interruption
10	F4	ADD	$D + M(R(3)) \rightarrow D$
11	30 0C	BR	branch. à 0C
13	70	RET	$M(R(2)) = (3.0) \rightarrow (X.P) ; R(2)+1.1 \rightarrow IE$: retour au P.P.
→ 14	22	DEC R(2)	
15	6A	INP 2	$W_i \rightarrow D$ et $W_i \rightarrow M(R(2)) = M(F7)$
16	61	OUT 1	Affichage W_i et $R(2) + 1$
17	22	DEC R(2)	$R(2) - 1$
18	78	SAV	$T = (3.0) \rightarrow M(R(2)) = M(F7)$
19	34 19	B1	Branchement à 19 si $EF_1 = 1$ soit $\overline{EF}_1 = 0$
1B	30 13	BR	Branchement à 13 pour retour au P.P.

Fig. 6

– les nombres W_1, W_2, \dots, W_n sont présentés successivement sur les clés D_7 à D_0 et sont affichés sur les afficheurs binaires ou hexadécimaux après pression sur le bouton poussoir.

– les résultats partiels, W_1 puis $W_1 + W_2$, puis $W_1 + W_2 + W_3, \dots$ sont affichés après le dernier terme W_i introduit par action sur un commutateur (par exemple le commutateur \overline{EF}_1 : si $\overline{EF}_1 = 0$, l'addition et l'affichage du résultat ne sont pas réa-

lisés, si $\overline{EF}_1 = 1$ l'addition du résultat partiel précédent et du dernier nombre W_i s'il a bien été introduit, et l'affichage du résultat se réalisent.

En résumé, à la mise en route (\overline{CLEAR} passe à 1) les afficheurs indiquent 0. \overline{EF}_1 est dans sa position de repos $\overline{EF}_1 = 0$. W_1 est présenté sur les clés. Il doit s'afficher après pression sur le bouton poussoir. Le passage de \overline{EF}_1 à 1 permet l'addition avec le total partiel précédent (c'est-à-dire 0) et l'affichage du résultat W_1 . W_2 est présenté sur les clés. Il est affiché après pression sur le bouton poussoir. Le passage de \overline{EF}_1 à 1 permet l'addition avec le total partiel précédent (soit W_1) et l'affichage du résultat $W_1 + W_2$ et ainsi de suite...

IV. 2. Solution proposée (voir organigramme et programme sur les figures 5 et 6)

Elle fait appel à l'instruction IDLE : l'utilisateur a ainsi tout son temps pour présenter le nombre W_i à ajouter au total partiel. Ce nombre W_i est chargé (en mémoire et sur le registre D) pendant le sous-programme d'interruption.

Un branchement, conditionné à l'état de EF_1 bloque le sous-programme et W_i reste affiché. Nous avons encore choisi le registre R (3) comme pointeur de données du programme principal.

Lors de l'essai éventuel de ce programme, le fonctionnement peut être troublé par les rebonds du commutateur \overline{EF}_1 . On obtient un fonctionnement correct en utilisant un cordon à la place du commutateur...

Le format du résultat est ici limité à 8 bits. Il n'est pas difficile de prévoir l'extension à 9 bits (utilisation de DF et de la sortie Q)

V. Application de l'interruption par le canal INTERRUPT : calculatrice 4 opérations

V. 1. But poursuivi

Nous nous proposons d'établir le programme d'une calculatrice 4 opérations dont le fonctionnement est le suivant :

- après mise en marche (\overline{CLEAR} passe à 1), le premier opérande W_1 présenté sur les clés est introduit en appuyant sur le bouton poussoir. Il est affiché pour vérification sur les LED et les afficheurs hexadécimaux
- un deuxième opérande W_2 est présenté sur les clés, est introduit en appuyant sur le bouton poussoir et est affiché.
- une opération sur les deux opérandes W_1 et W_2 peut être effectuée en positionnant correctement un commutateur affecté à l'opération désirée, puis en appuyant sur le bouton poussoir. Le résultat $W_1 \circ W_2$ est alors affiché et vient remplacer l'opérande W_1 .

Nous conviendrons d'affecter le commutateur \overline{EF}_1 à l'addition $W_1 + W_2$, le commutateur \overline{EF}_2 à la soustraction $W_1 - W_2$, le commutateur \overline{EF}_3 à la multiplication $W_1 \times W_2$ et le commutateur \overline{EF}_4 à la division $W_1 : W_2$. Pour choisir une opération parmi les quatre possibles, on met le \overline{EF} correspondant à 0.

- un troisième opérande est présenté sur les clés, est introduit en appuyant sur le bouton poussoir et est affiché.

- une opération entre le résultat précédent W_1 o W_2 (remplaçant W_1) et ce nouvel opérande W_3 (remplaçant W_2) est obtenu par la même méthode : choix de l'opération en mettant le \overline{EF} correspondant à 0 et commande d'exécution et d'affichage du résultat en appuyant sur le bouton poussoir. Et ainsi de suite avec d'autres opérandes introduits ultérieurement.

V. 2. Solution proposée

Elle est décrite par les organigrammes des figures 7.a à 7.f et par le programme de la figure 8. Voici néanmoins quelques éléments destinés à en faciliter la compréhension.

a) Le programme comporte :

- un programme principal dont le compteur est le registre R(0) et qui utilise le registre R(3) comme pointeur de données (fig. 7.a)

- quatre « sous-programmes » correspondant aux quatre opérations, pas totalement indépendants du programme principal car ils utilisent le même compteur et le même pointeur de données (le terme sous-programme est ici un peu abusif mais il est commode !). Il est fait appel à ces sous-programmes grâce à des instructions de branchement conditionnés à l'état des drapeaux. On retourne au programme principal grâce à un branchement inconditionnel (fig. 7.b à 7.e)

- un sous-programme d'interruption parvenant à l'issue d'une instruction IDLE laissant le temps à l'utilisateur, soit de présenter un opérande sur les clés, soit de placer une entrée \overline{EF} à 0 (fig. 7.f)

b) Le programme principal commence, comme à l'habitude, par une initialisation des registres utilisés. A l'issue d'une première mise en attente débouchant sur une interruption par le canal $\overline{INTERRUPT}$, le premier opérande W_1 est introduit. Cet opérande (au lancement du programme) ou un résultat partiel est affiché et le microprocesseur est à nouveau placé en état d'attente d'interruption.

L'interruption débouche, si aucune opération n'est effectuée (tous les \overline{EF} étant à 1), sur un affichage du nombre présent sur les clés et son chargement sur le registre D. Il faut remarquer qu'une décrémentation du registre R(3) avant stockage de ce nombre évite qu'il ne vienne se substituer en mémoire au résultat partiel. Ce résultat partiel est à nouveau pointé à l'issue de l'instruction de sortie qui provoque une incrémentation du registre R(3). Si une opération est demandée, elle utilise le résultat partiel précédent stocké à l'adresse R(3) et le nouvel opérande contenu dans le registre D. Le résultat vient remplacer en mémoire le résultat partiel précédent et est affiché.

c) Les sous-programmes d'addition et de soustraction sont élémentaires.

Le sous-programme de multiplication utilise la technique des additions successives. Le registre R(9) s'enrichit à chaque tour de boucle de M(R(3)). Le registre R(8) est utilisé comme compteur de boucles.

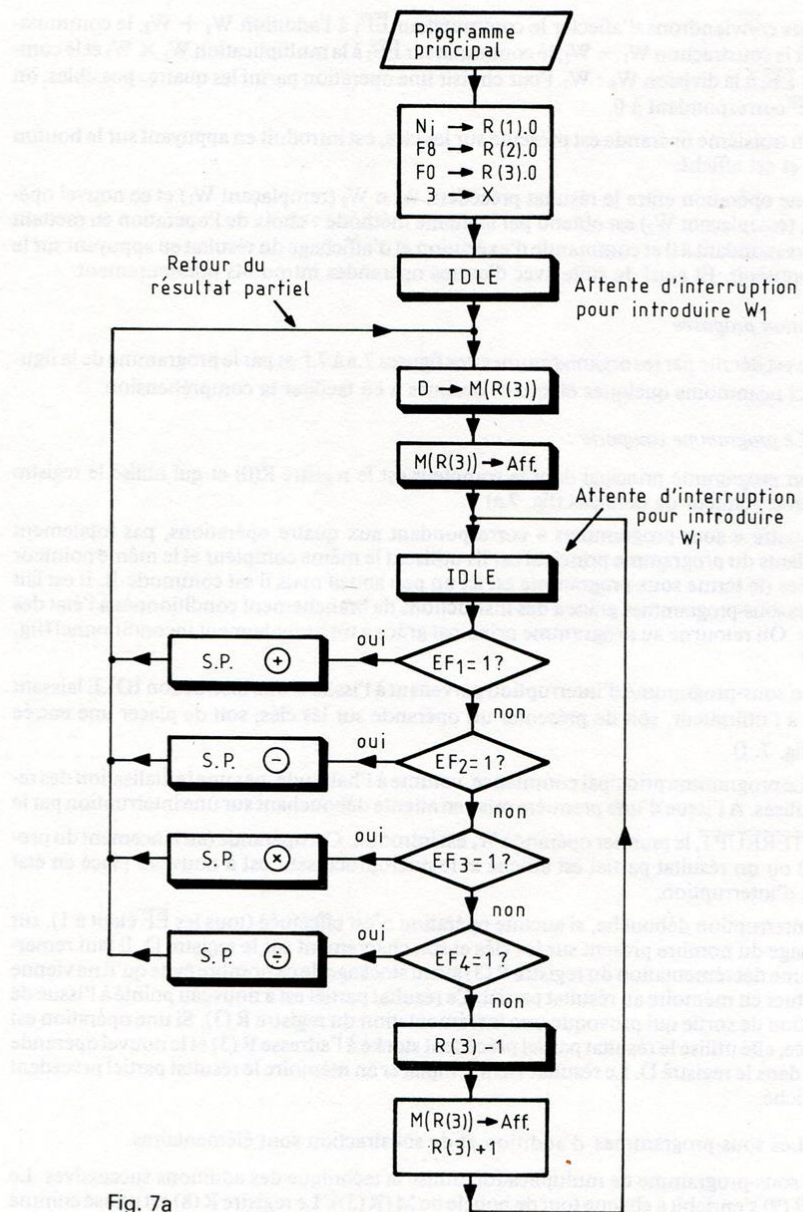


Fig. 7a

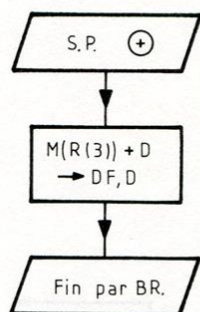


Fig. 7b

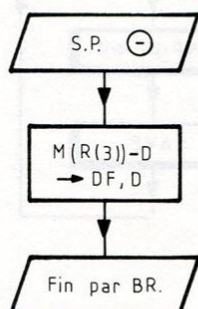


Fig. 7c

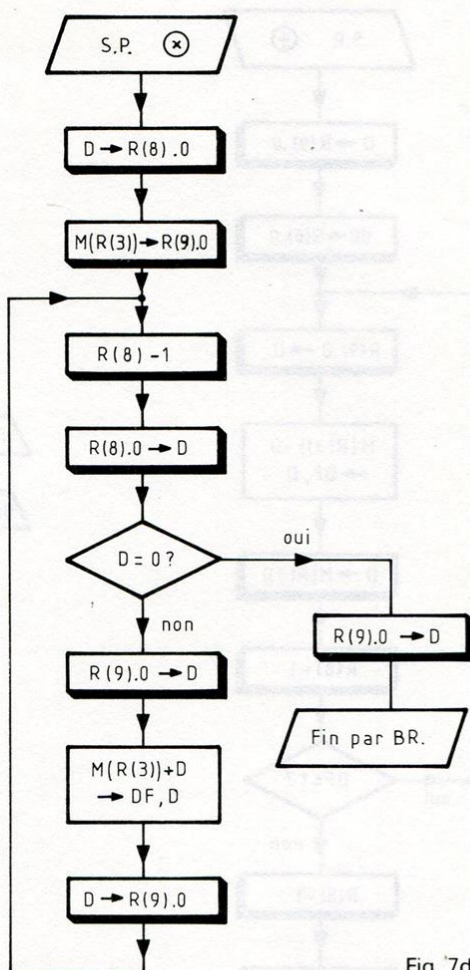


Fig. 7d

Le principe du sous-programme de division consiste à compter combien de fois W_i dernier opérande rentré est contenu dans $M(R(3))$ résultat partiel précédent. Le registre $R(8)$ joue le rôle de compteur de boucles décrites. A chaque tour $M(R(3))$ est diminué de W_i . On arrête lorsque DF passe à 1. Le résultat de la soustraction est alors négatif et l'on a opéré un tour de trop ce qui justifie la décrémentation effectuée à la fin sur le compteur de boucles

d) La seule instruction « active » du sous-programme d'interruption est l'instruction INPUT. Les autres instructions concernent la gestion du sous-programme lui-même.

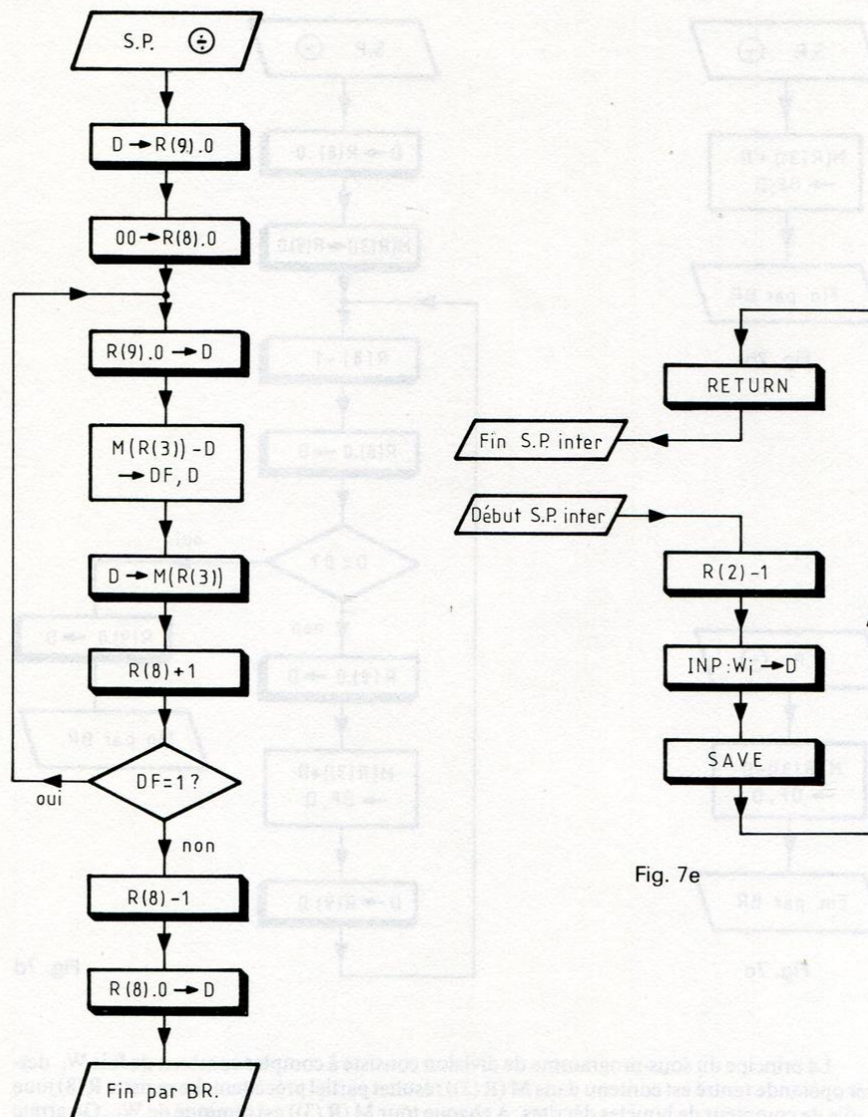


Fig. 7f

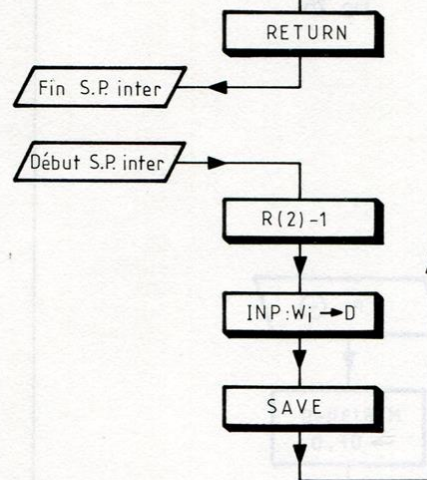


Fig. 7e

Adresse	Code et opérande	Mnémonique	Commentaire
00	F8 40	LDI	} $N_i = 40 \rightarrow R(1).0$
02	A1	PLO R(1)	
03	F8 F8	LDI	
05	A2	PLO R(2)	} $F8 \rightarrow R(2).0$
06	F8 F0	LDI	
08	A3	PLO R(3)	} $F0 \rightarrow R(3).0$
09	E3	SEX	
0A	00	IDL	attente d'interruption (pour rentrer W_1)
0B	53	STR	$D \rightarrow M(R(3)) = M(F0)$
0C	61	OUT 1	$M(R(3)) \rightarrow \text{afficheurs}; R(3) + 1$
0D	23	DEC R(3)	$R(3) - 1$
0E	00	IDL	attente d'interruption
0F	34 1C	B1	branch. à +
11	35 1E	B2	branch. à -
13	36 22	B3	branch. à X
15	37 31	B4	branch. à ÷
17	23	DEC R(3)	$R(3) - 1$
18	53	STR	$D \rightarrow M(R(3)) = M(EF)$
19	61	OUT 1	$M(R(3)) = M(EF) \rightarrow \text{afficheurs}; R(3) + 1$
1A	30 0E	BR	branch. avant opération
1C	F4	ADD	$M(R(3)) + D \rightarrow DF, D$
1D	30 0B	BR	branch. à stockage en vue d'affichage
1F	F5	SD	$M(R(3)) - D \rightarrow DF, D$
20	30 0B	BR	branch. à stockage en vue d'affichage
22	A8	PLO R(8)	$D \rightarrow R(8).0$
23	F0	LDX	$M(R(3)) \rightarrow D$
24	A9	PLO R(9)	$D \rightarrow R(9).0$
25	28	DEC R(8)	$R(8) - 1$
26	88	GLO R(8)	$R(8).0 \rightarrow D$
27	32 2E	BZ	branch. à $R(9).0 \rightarrow D$ si $D = 0$
29	89	GLO R(9)	$R(9).0 \rightarrow D$
2A	F4	ADD	$M(R(3)) + D \rightarrow DF, D$
2B	A9	PLO R(9)	$D \rightarrow R(9).0$
2C	30 25	BR	branch. à $R(8) - 1$
2E	89	GLO R(9)	$R(9).0 \rightarrow D$
2F	30 0B	BR	branch. à stockage en vue d'affichage
31	A9	PLO R(9)	$D \rightarrow R(9).0$
32	F8 00	LDI	} $00 \rightarrow R(8).0$
34	A8	PLO R(8)	
35	89	GLO R(9)	$R(9).0 \rightarrow D$
36	F5	SD	$M(R(3)) - D \rightarrow DF, D$
37	53	STR	$D \rightarrow M(R(3))$
38	18	INC R(8)	$R(8) + 1$
39	33 35	BR	branch. à $R(9) \rightarrow D$ si $DF = 1$
3B	28	DEC R(8)	$R(8) - 1$
3C	88	GLO R(8)	$R(8).0 \rightarrow D$
3D	30 0B	BR	branch. à stockage en vue d'affichage
3F	70	RET	$M(R(2)) = (3.0) \rightarrow (X.P); R(2) + 1$
→ 40	22	DEC R(2)	$R(2) - 1$
41	6A	INP 2	$W_i \rightarrow D; W_i \rightarrow M(R(2))$
42	78	SAV	$T = (3.0) \rightarrow M(R(2))$
43	30 3F		branch. à retour

programme
principal

S.P. ⊕

S.P. ⊖

S.P. ⊗

S.P. ⊕

S.P.
d'interruption

Fig. 8

Quatrième partie

INTERFAÇAGE ET APPLICATIONS

Chapitre 17

L'INTRODUCTION DE DONNÉES

I. La sélection de fonctions

I. 1. Principe

D'une façon générale, lorsqu'on demande au microprocesseur de faire quelque chose, cela peut être en fonction de l'état logique d'un certain nombre d'entrées. Prenons immédiatement un exemple concret : supposez que les clés D_7 à D_0 de la maquette A soit remplacées par un commutateur 8 positions comme l'indique la figure 1. Le mot de 8 bits présent sur le commutateur est constitué par sept « 1 » et un « 0 », la position du « 0 » correspondant au numéro du plot relié à la masse ; les LED servent seulement d'indicateur. Pour que le microprocesseur sache ce qu'il doit faire, il doit naturellement scruter le commutateur ; cette scrutation se limite ici à une simple lecture du mot présent sur le bus des données, et est effectuée par une instruction INPUT qui d'une part mémorise ce mot dans la mémoire et d'autre part le charge dans l'accumulateur.

Le contenu de l'accumulateur peut par suite être examiné en vue d'orienter le programme dans l'exécution de la fonction souhaitée.

I. 2. Exemple

Le possesseur de la maquette A et d'un petit haut parleur pourra essayer l'application amusante suivante : il s'agit de synthétiser un son dont la fréquence est sélectionnée par l'état logique des clés D_7 à D_0 . Chaque clé simule une touche. L'enfoncement de la touche T_i , correspondant à la clé D_i , doit provoquer la génération d'un son de fréquence f_i . Nous décidons arbitrairement que $D_i = 1$ veut dire touche T_i enfoncée.

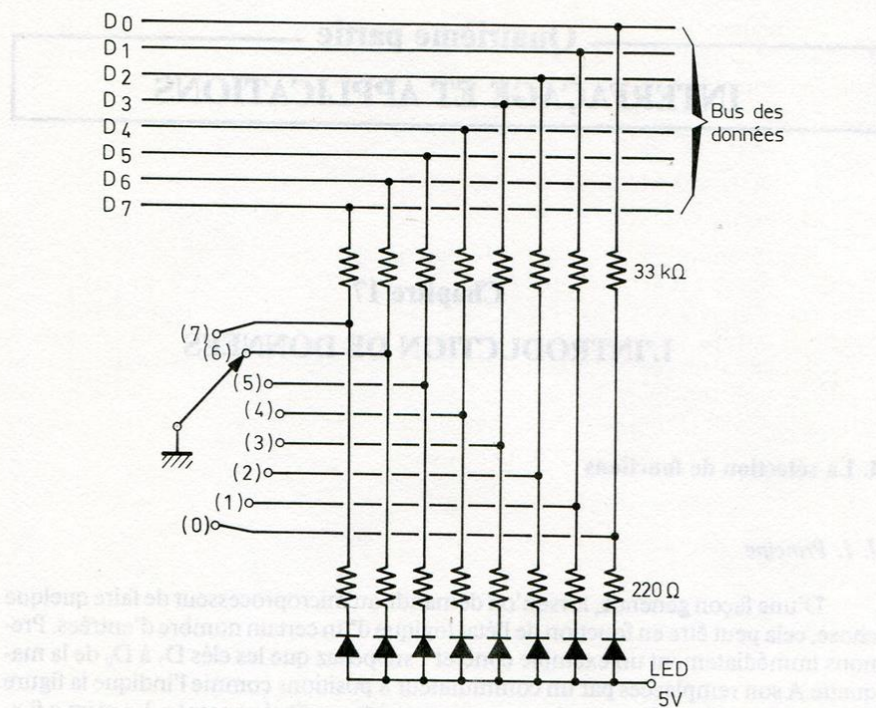


Fig. 1

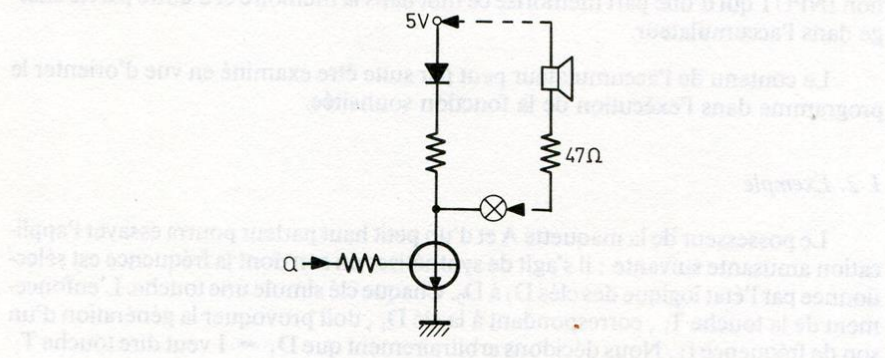


Fig. 2

Un haut parleur de petite puissance peut directement être relié à la douille de la maquette A correspondant à la sortie Q du microprocesseur (fig. 2). Pour obtenir un son de fréquence f , il faut réaliser la suite des opérations suivantes :

- Imposer Q au niveau 1 pendant $\frac{1}{2f}$ s.
- Imposer Q au niveau 0 pendant la même durée.
- Revenir en a).

Supposons que le programme de temporisation destiné à la génération du son, utilise le registre R (1).0 comme compteur de boucles. Comme nous avons 8 fréquences (8 notes) à synthétiser, le registre R (1).0 doit initialement contenir une valeur N parmi 8, cette valeur N dépendant de la touche enfoncée. Il est commode de ranger ces 8 valeurs de N dans une zone de la mémoire que nous appellerons « espace des notes ». La figure 3 montre que nous avons choisi R (2) comme pointeur de cet espace ; si R (2) pointe la note choisie, l'initialisation de R (1).0 se fait par la succession des deux opérations suivantes :

- Charger D par M (R (2))
- Transférer le contenu de D dans R (1).0

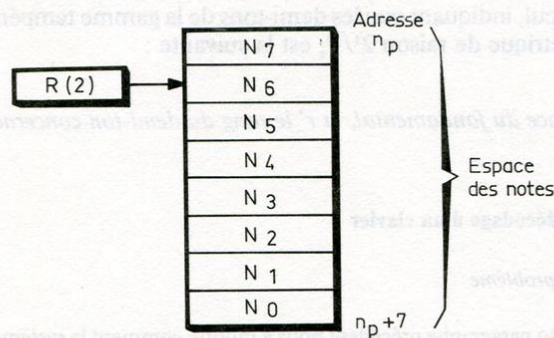


Fig. 3

La scrutation des 8 touches doit donc aboutir au pointage, par R (2), de la note souhaitée. L'organigramme de la figure 5 indique de quelle manière ceci est réalisé : R (2).0 étant initialisé à $n_p - 1$, chaque décalage à gauche du mot présent sur les touches a pour effet d'incrémenter R (2) ; lorsque la touche enfoncée est détectée par le test du drapeau DF, R (2) pointe la note correspondante.

La structure du programme est celle d'une boucle dans laquelle le clavier à 8 touches est constamment examiné. La durée d'exécution de cette boucle condition-

ne la fréquence du son généré. Si r est le rang de la touche (fig. 4) et si N est le contenu du registre R (1).0 exprimé en décimal, le lecteur pourra vérifier que la fréquence f de la note est donnée par la formule :

$$f = \frac{F}{96(r + N) + 304}$$

avec F = fréquence d'horloge du microprocesseur.

Touche	T_7	T_6	T_5	T_4	T_3	T_2	T_1	T_0
Rang r	1	2	3	4	5	6	7	8

Fig. 4

Nous donnons le programme complet de ce synthétiseur rudimentaire figure 6. Les valeurs de N ont été calculées de telle sorte que soient générés les 8 tons de la gamme majeure pour laquelle la fréquence du LA est 440 Hz, et ceci pour $F = 2$ MHz. Si on prend $F = 1$ MHz, on a à peu près l'octave inférieur. La formule utilisée pour le calcul, indiquant que les demi-tons de la gamme tempérée sont en progression géométrique de raison $2^{1/12}$, est la suivante :

$$f = f_0 \cdot 2^{r/12}$$

f_0 est la fréquence du fondamental, et r' le rang du demi-ton concerné (fig. 7).

II. Scrutation et décodage d'un clavier :

II. 1. Position du problème

L'exemple du paragraphe précédent nous a montré comment le système à microprocesseur était en mesure de prendre en compte l'état logique de 8 entrées. Si le nombre des entrées doit être plus important, on peut toujours les transmettre sur le bus des données par l'intermédiaire d'un multiplexeur. C'est d'ailleurs de cette façon que procède le microprocesseur lorsqu'il transmet sur le bus des adresses le contenu d'un registre R de 16 bits.

Une autre façon d'introduire des données est de le faire par l'intermédiaire d'un clavier. Si le clavier a une structure matricielle, il s'interface très simplement avec le microprocesseur, mais en contrepartie la programme chargé de le scruter et de le décoder occupe beaucoup d'espace mémoire.

A titre d'exemple, considérons le clavier à 4 lignes et 4 colonnes de la figure 8. Les lignes L_1 à L_4 sont respectivement les entrées \overline{EF}_1 à \overline{EF}_4 du microprocesseur ; si aucune touche n'est enfoncée, ces entrées sont au niveau logique 1 à cause des résistances R . Les colonnes C_1 à C_4 sont commandées par les 4 bits de poids fort du mot présent en sortie d'un coupleur CDP

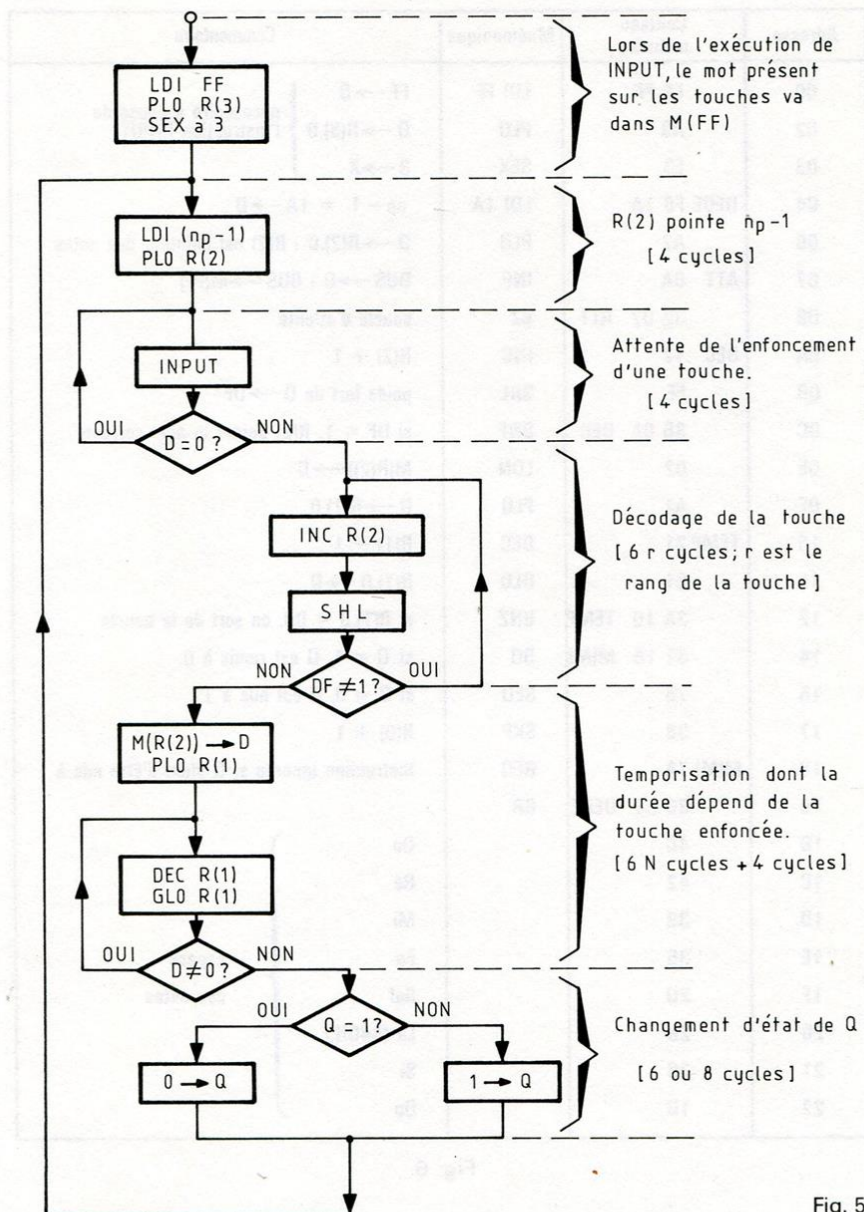


Fig. 5

Adresse	Contenu mémoire	Mnémonique	Commentaire
00	F8 FF	LDI FF	FF → D
02	A3	PLO	D → R(3).0
03	E3	SEX	3 → X
04	DEDE F8 1A	LDI 1A	np - 1 = 1A → D
06	A2	PLO	D → R(2).0 : R(2) est pointeur des notes
07	ATT 6A	INP	BUS → D ; BUS → M(FF)
08	32 07 ATT	BZ	boucle d'attente
0A	DEC 12	INC	R(2) + 1
0B	FE	SHL	poids fort de D → DF
0C	3B 0A DEC	BNF	si DF = 1, R(2) pointe la note correcte
0E	02	LDN	M(R(2)) → D
0F	A1	PLO	D → R(1).0
10	TEMP 21	DEC	R(1) - 1
11	81	GLO	R(1).0 → D
12	3A 10 TEMP	BNZ	si R(1).0 = 00, on sort de la boucle
14	31 18 MIMI	BQ	si Q = 1, Q est remis à 0.
16	7B	SEQ	si Q = 0, Q est mis à 1
17	38	SKP	R(0) + 1
18	MIMI 7A	REQ	Instruction ignorée si Q vient d'être mis à 1
19	30 04 DEDE	BR	
1B	4C		Do
1C	42		Ré
1D	39		Mi
1E	35		Fa
1F	2D		Sol
20	26		La (440H ₂)
21	20		Si
22	1D		Do

Fig. 6

note	do	do*	ré	ré*	mi	fa	fa*	sol	sol*	la	la*	si	do
r'	0	1	2	3	4	5	6	7	8	9	10	11	12
f(Hz)	262		294		330	349		392		440		494	523
N(Hexa)	4C		42		39	35		2D		26		20	1D

Fig. 7

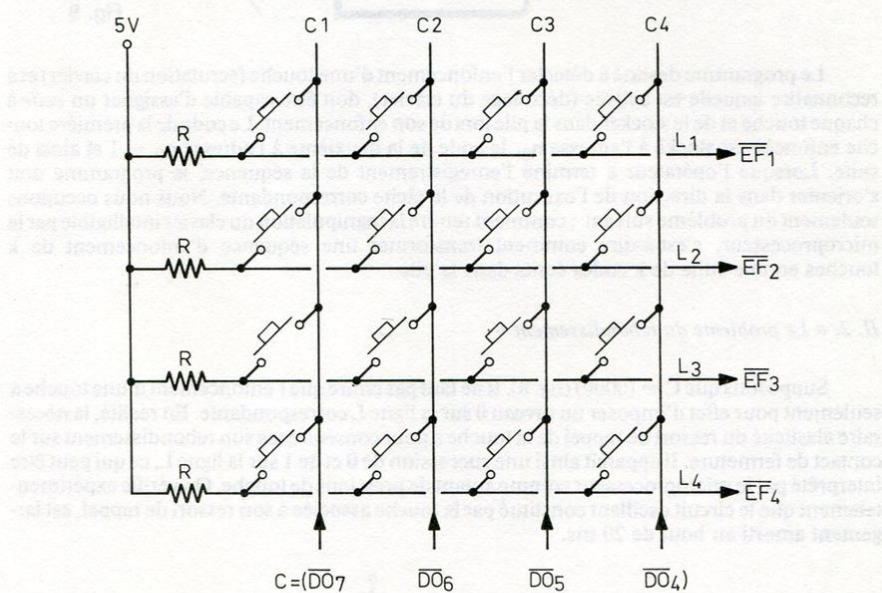


Fig. 8

1852 : si la maquette B est utilisée, il faut faire attention au fait que la douille correspondant à la sortie DO_i supporte en réalité le complément de DO_i , ce qui veut dire que le mot de 4 bits $C = (C_1 C_2 C_3 C_4) = (\overline{DO_7}, \overline{DO_6}, \overline{DO_5}, \overline{DO_4})$.

Chaque touche T_{ij} est représentée par ses coordonnées (i, j) : i est le numéro de la ligne, j celui de la colonne.

Le but est le suivant : le microprocesseur doit accomplir une tâche lorsqu'un certain nombre k de touches ont été enfoncées, tâche dépendant de la séquence que forment ces k touches. L'enfoncement d'une touche étant fugitif, il faut s'arranger pour que cette séquence soit enregistrée dans une zone de mémoire RAM que nous appellerons « pile ».

La figure 9 montre que nous avons choisi le registre R (6) comme pointeur de pile ; $np = FO$ est l'adresse du bas de la pile.

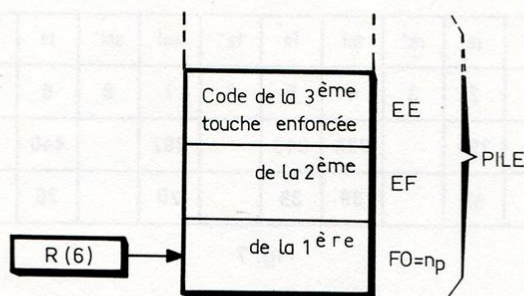


Fig. 9

Le programme destiné à détecter l'enfoncement d'une touche (scrutation du clavier) et à reconnaître laquelle est activée (décodage du clavier), doit être capable d'assigner un *code* à chaque touche et de le stocker dans la pile lors de son enfoncement. Le code de la première touche enfoncée est stocké à l'adresse n_p , le code de la deuxième à l'adresse $n_p - 1$ et ainsi de suite. Lorsque l'opérateur a terminé l'enregistrement de la séquence, le programme doit s'orienter dans la direction de l'exécution de la tâche correspondante. Nous nous occupons seulement du problème suivant : comment rendre la manipulation du clavier intelligible par le microprocesseur, c'est-à-dire comment transformer une séquence d'enfoncement de k touches en une suite de k codes écrits dans la pile.

II. 2. « Le problème du rebondissement »

Supposons que $C = (0000)$ (fig. 8). Il ne faut pas croire que l'enfoncement d'une touche a seulement pour effet d'imposer un niveau 0 sur la ligne L correspondante. En réalité, la nécessaire élasticité du ressort de rappel de la touche a pour conséquence son rebondissement sur le contact de fermeture. Il apparaît ainsi une succession de 0 et de 1 sur la ligne L , ce qui peut être interprété par le microprocesseur comme autant de pressions de touche. On vérifie expérimentalement que le circuit oscillant constitué par la touche associée à son ressort de rappel, est largement amorti au bout de 20 ms.

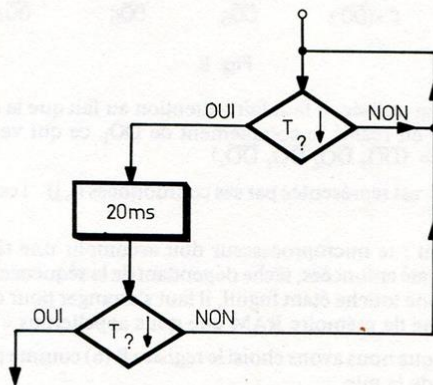


Fig. 10

L'organigramme de la figure 10 donne la solution au problème du rebondissement. Le symbole $T \downarrow$ veut dire « une touche est enfoncée ». Il faut remarquer qu'après la temporisation de 20 ms, le programme examine si la touche est toujours enfoncée ; si cela n'est pas le cas, c'est qu'en réalité il a été matériellement impossible qu'une touche ait été enfoncée, et par conséquent seul un signal parasite de durée inférieure à 20 ms a pu déclencher l'exécution de la temporisation. Notre programme anti-rebondissements est donc également destiné à « filtrer » les parasites de durée inférieure à 20 ms.

II. 3. Détection de la ligne

Pour décoder le clavier, c'est-à-dire pour reconnaître les coordonnées de la touche enfoncée, il est commode de commencer par détecter sur quelle ligne L se trouve cette touche. Le résultat (le numéro de la ligne) doit être stocké dans la pile sous la forme d'un *code* qu'on doit attribuer à chaque ligne. Le choix de ce code est déterminé par des considérations de simplicité du programme qui en résulte. Décidons que les 4 bits de poids faible de chaque ligne de la pile soient destinés à contenir :

- le code 0001 si la ligne L_1 est activée
- le code 0010 si la ligne L_2 est activée
- le code 0100 si la ligne L_3 est activée
- le code 1000 si la ligne L_4 est activée.

Le programme de détection de la ligne peut alors avoir la structure de la figure 11. Au cours de son exécution, les colonnes sont maintenues au niveau 0 grâce à l'instruction OUT-PUT ; cette instruction a pour effet de transférer le mot binaire (11110000) en sortie du coupleur CDP 1852 de la maquette B, à condition que ce mot soit au préalable contenu dans la mémoire. Nous avons choisi le registre R (5) comme pointeur de l'espace des données destinées à être sorties (fig. 12) ; FB est l'adresse du haut de cet espace, FF l'adresse du bas. Le résultat de l'enfoncement d'une touche est le stockage, dans la pile à l'adresse FO, du code de la ligne correspondante.

II. 4. Détection de la colonne :

Le principe de la détection de la colonne correspondant à la touche enfoncée consiste à faire circuler un niveau 0 successivement sur les colonnes C_1, C_2, C_3 puis C_4 en décrivant le processus suivant :

- a) $C = (0111)$ est placé sur les colonnes ; seul l'enfoncement d'une touche T_{11} est détecté
- b) si $T_{11} \downarrow$, la colonne est trouvée (c'est la colonne 1)
sinon $C = (1011)$ est placé sur les colonnes
- c) si $T_{12} \downarrow$, la colonne est trouvée (c'est la colonne 2)
sinon $C = (1101)$ est placé sur les colonnes
- d) si $T_{13} \downarrow$, la colonne est trouvée (c'est la colonne 3)
sinon $C = (1110)$ est placé sur les colonnes
- e) si $T_{14} \downarrow$, la colonne est trouvée (c'est la colonne 4)
sinon, cela veut dire qu'aucune touche n'est enfoncée ; alors $C = (0000)$ est placé sur les colonnes en vue de la détection de la ligne.

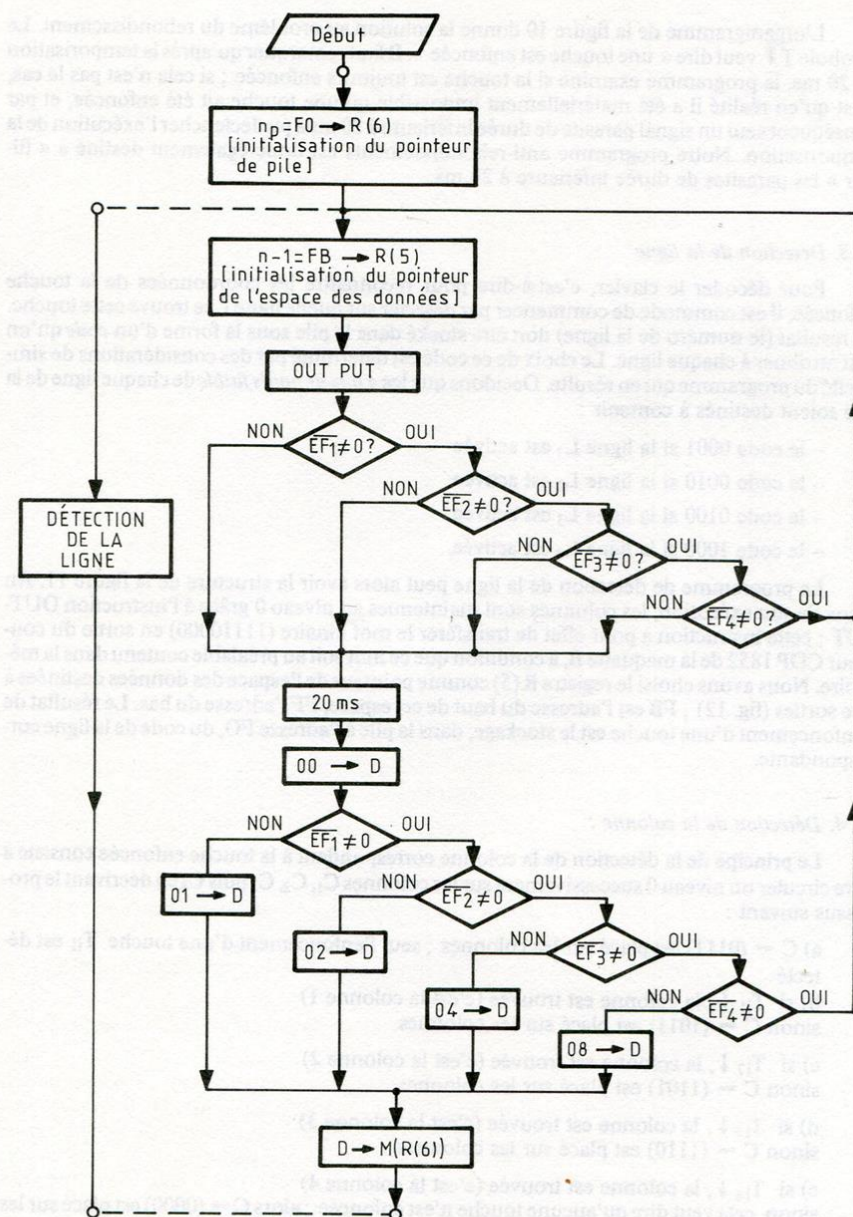


Fig. 11

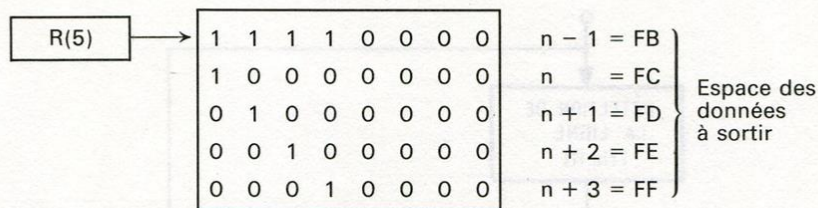


Fig. 12

On peut remarquer que le code à assigner à chaque colonne est tout trouvé. En effet, ce code peut être la valeur de C présente sur les colonnes lorsque la touche est détectée. En réalité, l'espace des données (fig. 12) contient les valeurs de C complémentées ; c'est celles-ci que nous choisirons comme code. Par conséquent, les 4 bits de poids fort des lignes d'adresses n à $n + 3$ de l'espace des données contiennent :

- le code 1000 si la colonne C_1 est activée
- le code 0100 si la colonne C_2 est activée
- le code 0010 si la colonne C_3 est activée
- le code 0001 si la colonne C_4 est activée

La structure du programme de détection de la colonne est indiquée figure 13. Le code de la touche, c'est-à-dire la réunion du code de la ligne et du code de la colonne, est finalement stocké dans la pile à l'adresse $F0$.

II. 5. Détection d'une séquence de k touches :

Si on désire que le microprocesseur exécute une certaine tâche seulement après l'enfoncement de k touches, il est nécessaire d'organiser la procédure suivante :

- a) après le stockage dans la pile, à l'adresse n_p , du code de la première touche enfoncée, il faut décoder la deuxième touche. Ceci ne doit être réalisé qu'après relâchement de la première touche.
- b) après le stockage dans la pile, à l'adresse $n_p - 1$, du code de la deuxième touche enfoncée, il faut décoder la troisième touche.
- c) lorsque la séquence des k touches est stockée dans la pile, le programme doit s'orienter dans la direction d'exécution de la tâche à remplir.

La figure 14 explicite le processus de stockage, dans la pile, d'une séquence d'enfoncement de k touches.

A titre d'exemple, nous donnons figure 15 un programme permettant de stocker dans la pile la succession des codes correspondant à l'enfoncement de 6 touches. On peut essayer ce programme sans disposer de clavier, à condition de simuler une pression de touche par un contact momentané entre l'une des 4 sorties du coupleur de la maquette B et l'une des 4 entrées « drapeau » de la maquette A. Comme le programme de détection de la colonne utilise l'instruction OUTPUT, le code de la touche apparaît sur les LED de la maquette B tant que la touche reste enfoncée.

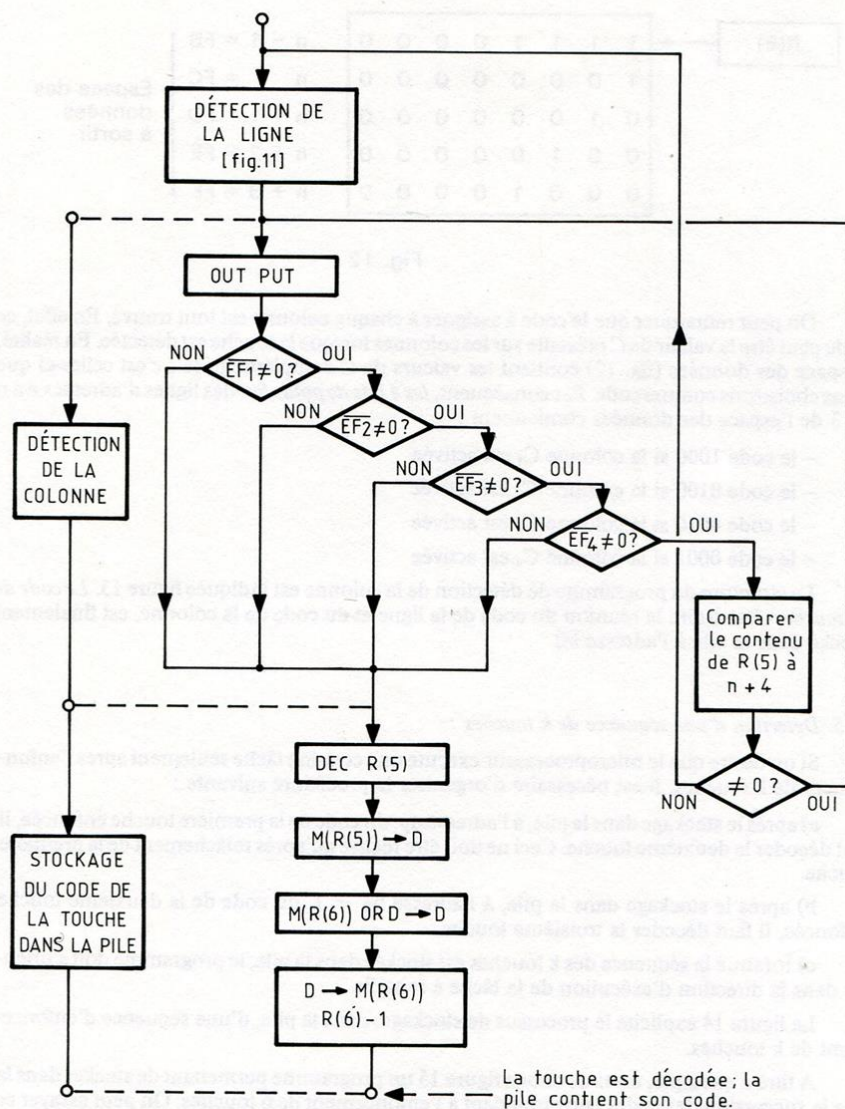


Fig. 13

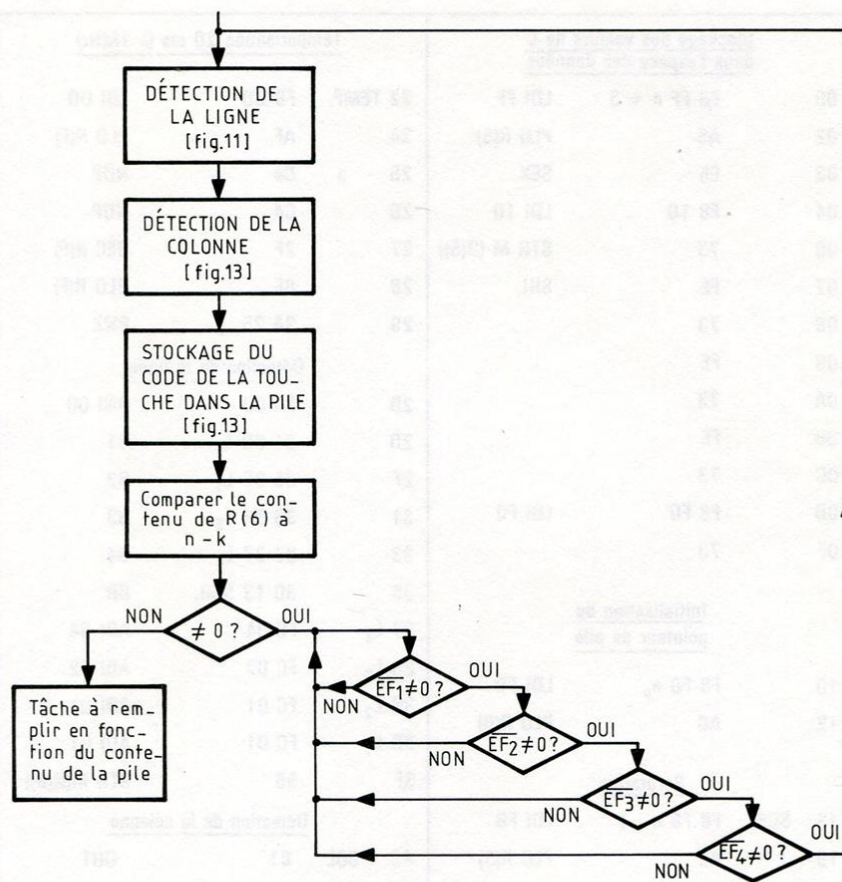


Fig. 14

Stockage des valeurs de C dans l'espace des données			Temporisation 20 ms (à 1MHz)		
00	F8 FF n + 3	LDI FF	22 TEMP.	F8 D0	LDI D0
02	A5	PLO R(5)	24	AF	PLO R(F)
03	E5	SEX	25 p	C4	NOP
04	F8 10	LDI 10	26	C4	NOP
06	73	STR M (R(5))	27	2F	DEC R(F)
07	FE	SHL	28	8F	GLO R(F)
08	73		29	3A 25 p	BNZ
09	FE		Détection de la ligne		
0A	73		2B	FA 00	ANI 00
0B	FE		2D	34 3D L ₁	B1
0C	73		2F	35 3B L ₂	B2
0D	F8 F0	LDI F0	31	36 39 L ₃	B3
0F	73		33	37 37 L ₄	B4
Initialisation du pointeur de pile			35	30 13 SCR.	BR
10	F8 F0 n _p	LDI F0	37 L ₄	FC 04	ADI 04
12	A6	PLO R(6)	39 L ₃	FC 02	ADI 02
Scrutation			3B L ₂	FC 01	ADI 01
13 SCR.	F8 FB n - 1	LDI FB	3D L ₁	FC 01	ADI 01
15	A5	PLO R(5)	3F	56	STR M(R(6))
16	E5	SEX	Détection de la colonne		
17	61	OUT	40 COL	61	OUT
18	34 22 TEMP.	B1	41	34 4E STR	B1
1A	35 22 TEMP.	B2	43	35 4E STR	B2
1C	36 22 TEMP.	B3	45	36 4E STR	B3
1E	37 22 TEMP.	B4	47	37 4E STR	B4
20	30 13 SCR.	BR	49	85	GLO R(5)
			4A	3A 40 COL	BNZ
			4C	30 13 SCR	BR

Fig. 15

<u>Stockage dans la pile</u>			
4E	STR	25	DEC R(5)
4F		05	LD M(R(5))
50		E6	SEX
51		F1	OR M(R(6))
52		73	STXD
<u>Les k (6) touches ont-elles été enfoncées ?</u>			
53		86	GLO R(6)
54		FD EA $n_p - 6$	SDI EA
56		32 62 GO	BZ
58	REL	34 58 REL	B1
5A		35 58 REL	B2
5C		36 58 REL	B3
5E		37 58 REL	B4
60		30 13 SCR	BR
<u>Tâche à remplir</u>			
62	GO	A définir par le lecteur	

Fig. 15 (suite)

Chapitre 18

L'AFFICHAGE NUMÉRIQUE

I. La commande d'un afficheur numérique

Considérons un afficheur à 7 segments du type à *cathode commune*. Les cathodes de chacune des LED qui constituent un segment sont reliées entre elles et forment le point K. (fig. 1). Les 7 anodes sont commandées séparément par les signaux notés de a à g.

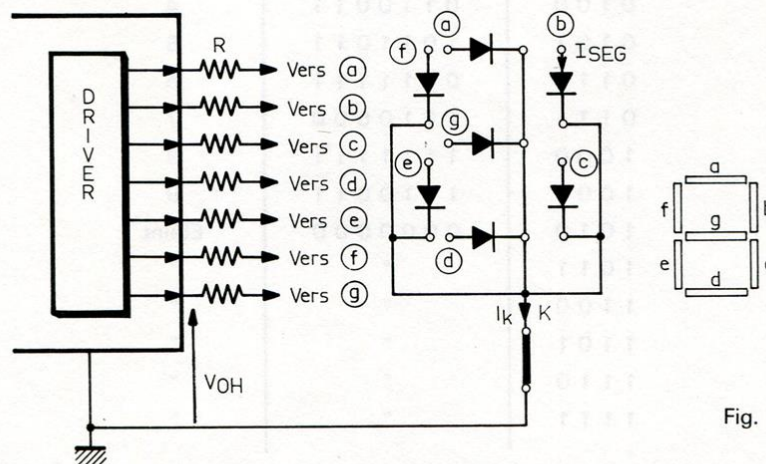


Fig. 1

Pour qu'un segment s'allume, il faut que la LED correspondante soit parcourue par un courant suffisant. Ce courant, typiquement de l'ordre de 20 mA pour des afficheurs classiques, doit être fourni par un transistor qui peut être monté en élément discret ou bien être partie constituante de l'étage « driver » d'un circuit intégré spécialement adapté à cette fonction. Si V_{OH} est la tension de sortie au niveau haut de cet étage driver, V_{DF} la chute de tension aux bornes de la LED allumée et I_{SEG} le courant qui la parcourt, il faut choisir $R = \frac{V_{OH} - V_{DF}}{I_{SEG}}$

Remarquez que le courant I_K qui circule dans la ligne cathode commune est 7 fois plus important que I_{SEG} si le caractère 8 est affiché.

Les 7 segments autorisent l'affichage de $2^7 = 128$ caractères différents ; parmi ceux-ci, un grand nombre n'ont pas de sens. Intéressons nous seulement aux caractères constitués par les chiffres de 0 à 9 : il est nécessaire et suffisant de disposer de 4 bits (notés DCBA) pour sélectionner l'un de ces 10 chiffres. Si la représentation décimale du mot binaire (DCBA) correspond précisément au caractère affiché, son décodage devra être confié à un décodeur à 4 entrées et 7 sorties répondant à la table de vérité de la figure 2.

D C B A	a b c d e f g	Caractère
0 0 0 0	1 1 1 1 1 1 0	0
0 0 0 1	0 1 1 0 0 0 0	1
0 0 1 0	1 1 0 1 1 0 1	2
0 0 1 1	1 1 1 1 0 0 1	3
0 1 0 0	0 1 1 0 0 1 1	4
0 1 0 1	1 0 1 1 0 1 1	5
0 1 1 0	0 0 1 1 1 1 1	6
0 1 1 1	1 1 1 0 0 0 0	7
1 0 0 0	1 1 1 1 1 1 1	8
1 0 0 1	1 1 1 0 0 1 1	9
1 0 1 0	0 0 0 0 0 0 0	Eteint
1 0 1 1	"	"
1 1 0 0	"	"
1 1 0 1	"	"
1 1 1 0	"	"
1 1 1 1	"	"

Fig. 2

Le circuit CD4511 de chez RCA remplit les fonctions d'un tel décodeur, ce décodeur est suivi d'un étage driver capable de fournir 25 mA en régime continu à chaque segment. Le CD4511 contient par ailleurs un registre latch (fig. 3) permettant de mémoriser le code DCBA lors du front montant de son entrée de déclenchement LE (Latch Enable) :

- si $LE = 0$, le latch est transparent
- si $LE = 1$, le latch contient le code présenté sur ses entrées lorsque LE était au niveau 0.

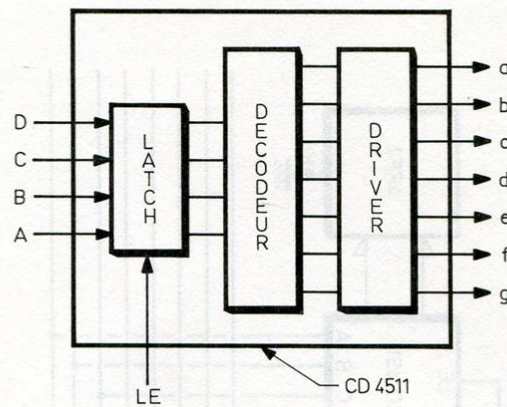


Fig. 3

La présence du registre latch incorporé au circuit CD 4511 suggère qu'il est relativement simple de réaliser un circuit d'« interface » entre la maquette A et un couple d'afficheurs 7 segments. Son schéma est donné figure 4 :

- l'ensemble des deux codes DCBA est présenté sur le bus des données.
- lors de l'exécution de l'instruction de sortie de code 61, N_0 est au niveau logique 1 et par conséquent $LE = \overline{TPB}$:
- au début du cycle d'exécution, $TPB = 0$ donc $LE = 1$; puis le mot $M(R(X))$ pointé par le registre $R(X)$ est transféré sur le bus des données
- lorsque TPB passe au niveau 1 ($LE = 0$), les latches sont transparents au mot $M(R(X))$ d'où il résulte l'affichage des deux caractères
- lorsque TPB revient au niveau 0 ($LE = 1$), les registres latch conservent en mémoire les codes des caractères, ce qui maintient leur affichage.

Compte tenu de la structure matérielle de ce circuit d'interface, le programme consistant à afficher le nombre $\square_1 \square_2$ (\square_1 est le caractère pour l'afficheur 1 et \square_2 le caractère pour l'afficheur 2) dont le code est contenu dans la mémoire à l'adresse pointée par $R(X)$, se limite à l'instruction OUT 1.

Supposons maintenant que nous désirions afficher un nombre de 4 chiffres. Nous pouvons adopter la structure du circuit d'interface précédent mais cela nécessite l'utilisation de 4 composants CD4511 ce qui est peu économique ; cette lourdeur matérielle est cependant compensée par l'extrême simplicité du programme d'affichage qui en résulte. En effet, numérotons les afficheurs de 1 à 4 et décidons que N_0 commande les registres latch pour les afficheurs 1 et 2, et N_1 ceux des afficheurs 3 et

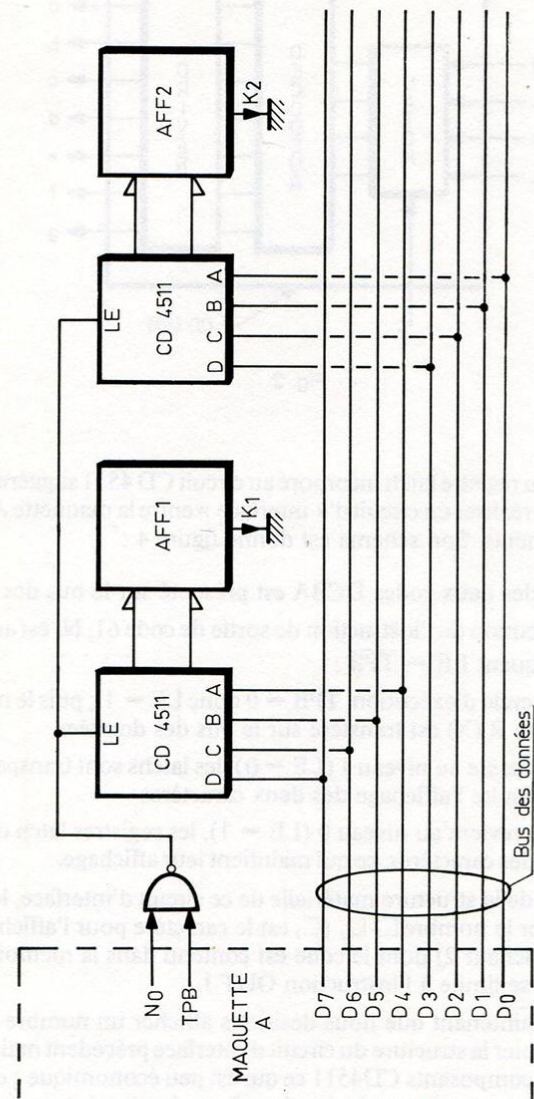


Fig. 4

4 : si les lignes mémoire d'adresses $R(X)$ et $R(X) + 1$ contiennent respectivement les codes des caractères \square_1 , \square_2 et \square_3 , \square_4 , le programme d'affichage de ces 4 caractères se limite à la succession des instructions OUT 1 et OUT 2.

En réalité, on peut notablement réduire la complexité d'un circuit d'interface destiné à l'affichage d'un nombre de 4 chiffres en adoptant la technique du multiplexage. Bien que cette solution aboutisse à l'élaboration d'un programme de structure moins évidente, nous la développerons dans les lignes ci-après car elle est aisément généralisable à un nombre quelconque d'afficheurs.

II. L'affichage numérique « multiplexé »

II. 1. Principe et interfacement avec le microprocesseur

Reportons nous à la figure 5. Nous n'avons représenté que deux segments par afficheur mais les 4 anodes d'un numéro d'ordre quelconque donné sont reliées entre elles : un seul circuit CD 4511 commande les 7 segments des 4 afficheurs.

Si les cathodes communes K_1, K_2, K_3, K_4 étaient reliées comme précédemment à la masse, il ne serait pas possible d'afficher un nombre de 4 chiffres constitué par des chiffres différents. Pour pouvoir contrôler l'allumage d'un afficheur, il faut être en mesure d'autoriser ou d'interdire le passage du courant dans la cathode commu-

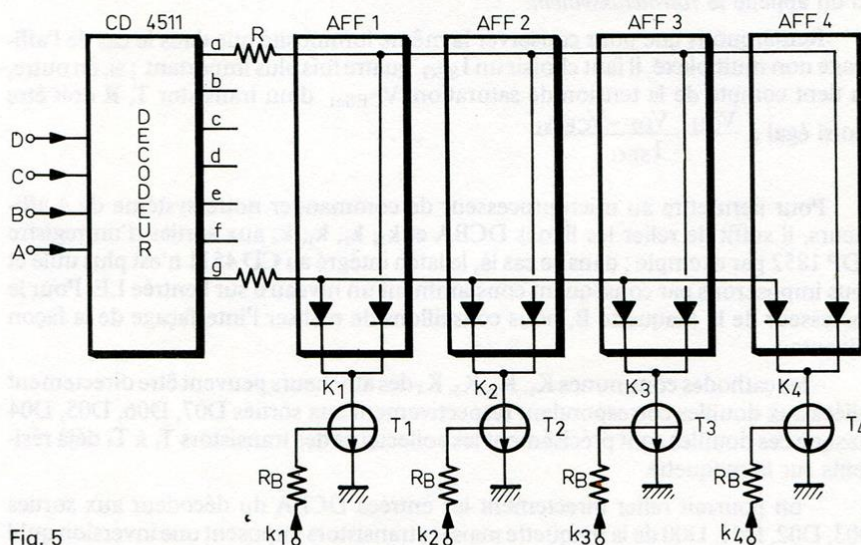


Fig. 5

ne correspondante ; ceci est réalisé par l'intermédiaire des transistors T_1, T_2, T_3, T_4 commandés respectivement par les signaux k_1, k_2, k_3, k_4 .

Pour afficher le mot $\square_1 \square_2 \square_3 \square_4$ la procédure à adopter est la suivante :

a) Sélectionner l'afficheur 1 en envoyant sur les bases des transistors la combinaison $(k_1, k_2, k_3, k_4) = (1000)$, puis présenter sur les entrées DCBA du décodeur le code du caractère \square_1 : \square_1 est affiché sur l'afficheur 1 pendant que les autres afficheurs sont éteints.

b) Sélectionner l'afficheur 2 en envoyant sur les bases des transistors la combinaison $(k_1, k_2, k_3, k_4) = (0100)$, puis présenter sur les entrées DCBA le code du caractère \square_2 : \square_2 est affiché sur l'afficheur 2 pendant que les autres afficheurs sont éteints.

c) Sélectionner l'afficheur 3, c'est-à-dire faire $(k_1, k_2, k_3, k_4) = (0010)$ puis présenter le code du caractère \square_3 sur les entrées DCBA.

d) Sélectionner l'afficheur 4, c'est-à-dire faire $(k_1, k_2, k_3, k_4) = (0001)$ puis présenter le code du caractère \square_4 sur les entrées DCBA.

e) Revenir en a).

L'affichage du mot $\square_1 \square_2 \square_3 \square_4$ consiste donc en l'affichage *successif* des caractères $\square_1, \square_2, \square_3, \square_4$ sur les afficheurs respectivement 1, 2, 3, 4. Si l'intervalle de temps entre l'affichage d'un caractère et le suivant est suffisamment court (Typiquement de 4 à 5 ms), l'œil a l'illusion d'un affichage permanent. Il est par ailleurs nécessaire de renouveler périodiquement l'affichage successif des caractères \square_1 à \square_4 : il s'agit de ce qu'on appelle le *rafraîchissement*.

Remarquons que pour conserver la même luminosité que dans le cas de l'affichage non multiplexé, il faut choisir un I_{SEG} quatre fois plus important ; si, en outre, on tient compte de la tension de saturation V_{CESat} d'un transistor T, R doit être choisi égal à $\frac{V_{OH} - V_{DF} - V_{CESat}}{I_{SEG}}$

Pour permettre au microprocesseur de commander notre système de 4 afficheurs, il suffit de relier les lignes DCBA et k_1, k_2, k_3, k_4 aux sorties d'un registre CDP 1852 par exemple ; dans ce cas là, le latch intégré au CD 4511 n'est plus utile et nous imposerons par conséquent constamment un niveau 0 sur l'entrée LE. Pour le possesseur de la maquette B, nous conseillons de réaliser l'interfaçage de la façon suivante :

- les cathodes communes K_1, K_2, K_3, K_4 des afficheurs peuvent être directement reliées aux douilles correspondant respectivement aux sorties D07, D06, D05, D04 puisque ces douilles sont précisément les collecteurs des transistors T_1 à T_4 déjà résidents sur la maquette.

- on pourrait relier directement les entrées DCBA du décodeur aux sorties D03, D02, D01, D00 de la maquette mais les transistors imposent une inversion qu'il

est préférable de compenser en insérant 4 inverseurs comme l'indique la figure 6. En effet, supposons que nous désirions afficher le caractère 8 sur l'afficheur 1 : si nous ne mettons pas d'inverseurs, le mot à présenter sur le bus des données est 87 (en hexadécimal) ; les 4 bits de poids faible de ce mot, soit 0111, doivent représenter le complément du nombre affiché, ce qui peut induire des complications dans la programmation.

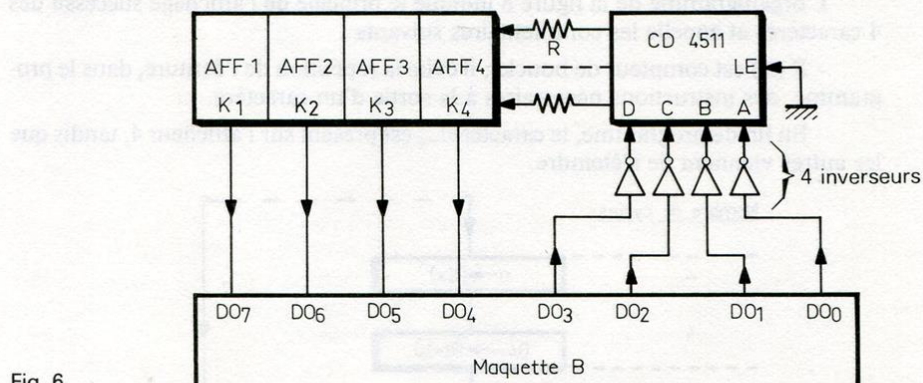


Fig. 6

II. 2. Programme d'affichage

Pour toute la suite, nous supposons réalisé le circuit d'interface de la figure 6.

Supposons que les 4 lignes mémoire d'adresses n , $n + 01$, $n + 02$, $n + 03$ contiennent respectivement les octets 8 \square_1 , 4 \square_2 , 2 \square_3 , 1 \square_4 (voir fig. 7). Nous conviendrons que les demi-octets \square_1 , \square_2 , \square_3 , \square_4 , sont les *codes* des caractères de 1 à 4.

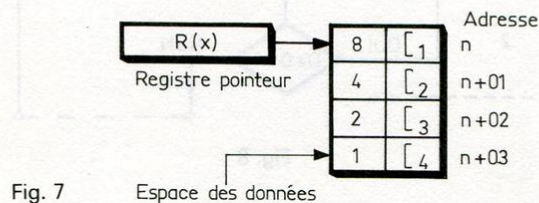


Fig. 7

L'ensemble de ces 4 lignes mémoire constitue l'*espace des données* et est pointé par le registre $R(X)$. Remarquons que, quels que soient les caractères à afficher, les 4 bits de poids fort du contenu de l'espace des données constituent nécessairement, en notation hexadécimale, la suite 8, 4, 2, 1.

L'instruction de base d'un programme d'affichage est l'instruction OUTPUT ; rappelons que lors de son exécution, l'opération suivante est effectuée :

$$M(R(X)) \rightarrow \text{BUS} ; R(X) + 1$$

Par conséquent, $R(X)$ étant initialisé à n , une succession de 4 instructions OUTPUT a pour effet d'afficher \square_1 sur l'afficheur 1, puis \square_2 sur l'afficheur 2, etc...

L'organigramme de la figure 8 indique le principe de l'affichage successif des 4 caractères et appelle les commentaires suivants :

- $R(X)$ est compteur de boucle ; il évite la répétition de l'écriture, dans le programme, des instructions nécessaires à la sortie d'un caractère.
- En fin de programme, le caractère \square_4 est présent sur l'afficheur 4, tandis que les autres viennent de s'éteindre.

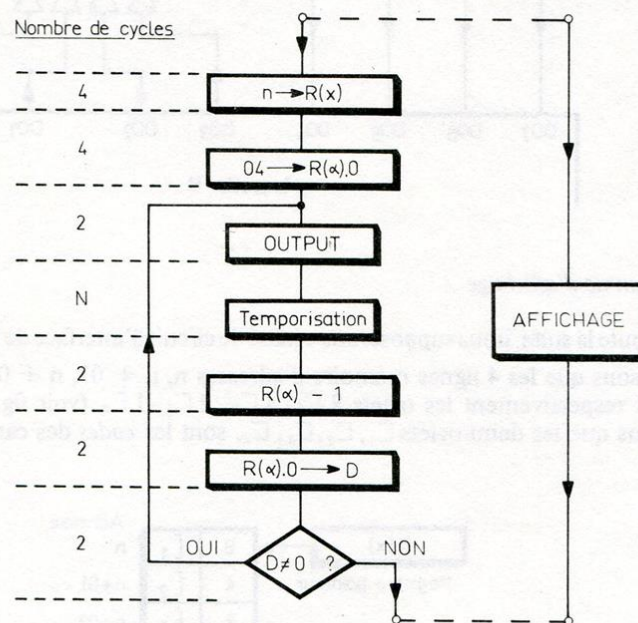


Fig. 8

Pour que l'œil ait cependant l'illusion de la présence des caractères $\square_1, \square_2, \square_3$, il est nécessaire de rafraîchir rapidement l'affichage en rebouclant le programme sur lui-même (fig. 9). En réalité, lors de la réalisation d'un projet déterminé, le microprocesseur n'est pas seulement chargé d'afficher un nombre, mais aussi d'exécuter un certain nombre de tâches annexes ayant pour rôle de modifier le contenu de l'espace

des données en vue précisément de l'affichage d'un nombre différent ; ces tâches annexes doivent nécessairement se terminer par un rafraîchissement de l'affichage (fig. 10). Compte-tenu de ce que nous venons de dire, quel est le rôle de la temporisation qui apparaît après l'instruction OUTPUT ?

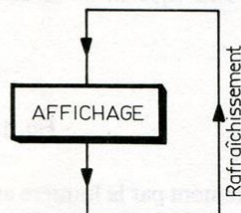


Fig. 9

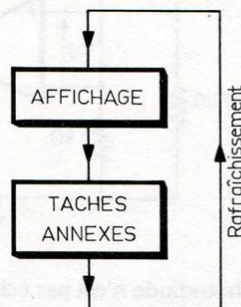


Fig. 10

Si N est le nombre de cycles requis pour l'exécution de la temporisation, et N_T celui requis pour l'exécution des tâches annexes, l'examen de la figure 8 et de la figure 10, nous indique que :

- les caractères $\square_1, \square_2, \square_3$ sont chacun présents sur les afficheurs pendant une durée correspondant à $N + 8$ cycles.
- le caractère \square_4 est présent pendant une durée correspondant à $N + N_T + 18$ cycles.

Par conséquent, si l'on désire une répartition à peu près uniforme de l'intensité lumineuse, il est nécessaire de respecter la condition $N \gg N_T$ ce qui impose de faire appel à un sous-programme de temporisation.

III. Application au comptage

III. 1. But proposé

Pour que le lecteur intéressé puisse étudier avec agrément les problèmes posés par la mise au point d'un programme d'affichage, nous proposons l'exemple ni trop simple ni trop compliqué suivant :

Il s'agit de compter le nombre de personnes (ou d'objets) traversant un faisceau lumineux. Ce faisceau lumineux commande la conductivité d'une photodiode qu'on peut insérer dans le montage de la figure 11 :

- si la photodiode est éclairée, la tension différentielle e_d est positive et la sortie du comparateur fournit un niveau logique 1.

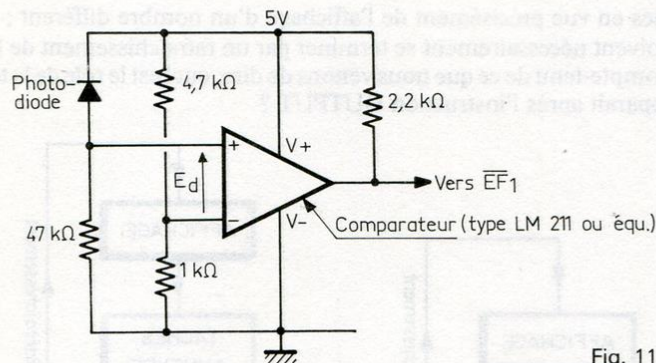


Fig. 11

– si la photodiode n'est pas éclairée (ou seulement par la lumière ambiante) e_d est négative et la sortie du comparateur fournit un niveau logique 0.

La sortie du comparateur est directement reliée à l'entrée \overline{EF}_1 du microprocesseur et nous constatons que $\overline{EF}_1 = 0$ tant que le faisceau est coupé.

Nous désirons que :

- lors de la mise en route, l'affichage soit 0000.
- à chaque coupure du faisceau lumineux, 1 soit additionné au nombre affiché.

La structure générale du programme est représentée figure 12.

Tant que le faisceau n'est pas coupé ($\overline{EF}_1 \neq 0$), le microprocesseur affiche le nombre dont la représentation est contenue dans l'espace des données. Dès que le faisceau est coupé, ce nombre est incrémenté puis immédiatement affiché. Le faisceau étant toujours coupé ($\overline{EF}_1 = 0$), le microprocesseur attend la sortie de la personne (ou de l'objet) du faisceau, tout en exécutant l'opération d'affichage.

III. 2. Les sous-programmes d'affichage et de temporisation

Le programme principal donné figure 12 indique que le rectangle « AFF » est utilisé deux fois. Il est donc commode de le considérer comme un sous-programme ; voyez figure 13 ce sous-programme donné en langage assembleur (c'est-à-dire que les instructions sont représentées par leur mnémonique). Vous observez que ce sous-programme d'affichage appelle lui-même un sous-programme de temporisation donné figure 14 et pour lequel $R(\beta)$ est compteur de boucles. Le nombre N définit la durée θ d'exécution du sous-programme de temporisation : si T est la période d'horloge et si N est grand devant 1, on a $\theta = 48 N.T$ (N est exprimé en décimal évidemment dans cette formule) ; pour obtenir une durée θ environ égale à 1 ms, il faut choisir $N = 14$ en hexadécimal, d'où il résulte une durée de $4 \times \theta$ soit 4 ms pour l'affichage (ce calcul est effectué pour $T = 1 \mu s$).

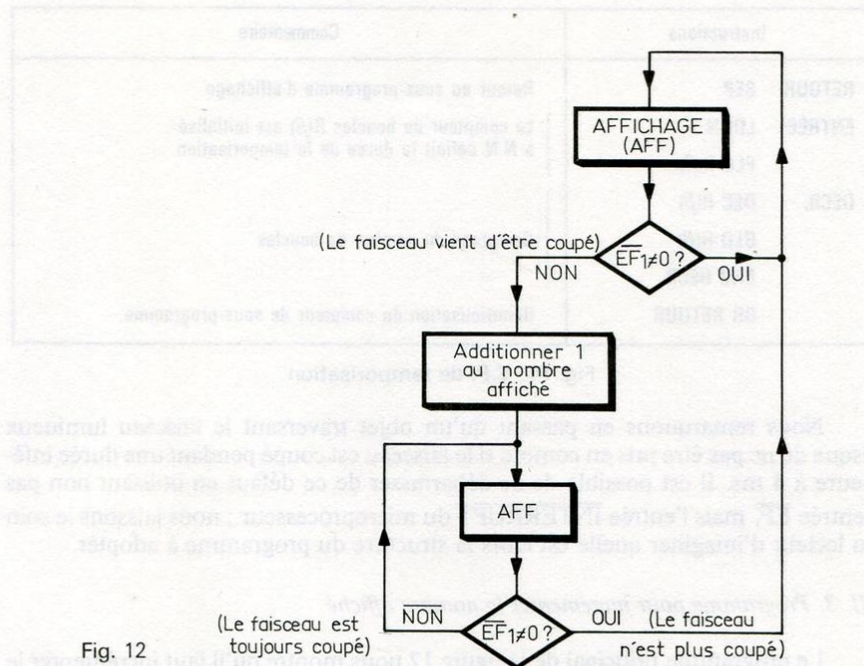


Fig. 12

Instructions		Commentaire
RETOUR	SEP	Retour au programme principal
ENTRÉE	LDI n	R(X) est initialisé à n, adresse du haut de l'espace des données.
	PLO R(X)	
	LDI 04	
OUTPUT	PLO R(x)	Le compteur de boucles R(x) est initialisé à 04
	SEX	
	OUT	Définition du registre pointeur.
	SEP	M (R(X)) → Bus des données.
	DEC R(x)	Appel du Sous-programme de temporisation
	GLO R(x)	Comptage du nombre de boucles
	BNZ OUTPUT	
	BR RETOUR	Réinitialisation du compteur de sous-programme

Fig. 13 : S.P. d'affichage

Instructions		Commentaire
RETOUR	SEP	Retour au sous-programme d'affichage
ENTRÉE	LDI N	} Le compteur de boucles $R(\beta)$ est initialisé à N.N définit la durée de la temporisation
	PLO $R(\beta)$	
DECR.	DEC $R(\beta)$	} Comptage du nombre de boucles
	GLO $R(\beta)$	
	BNZ DECR.	
	BR RETOUR	Réinitialisation du compteur de sous-programme.

Fig. 14 : S.P. de temporisation

Nous remarquons en passant qu'un objet traversant le faisceau lumineux risque de ne pas être pris en compte si le faisceau est coupé pendant une durée inférieure à 4 ms. Il est possible de se débarrasser de ce défaut en utilisant non pas l'entrée \overline{EF}_1 mais l'entrée $\overline{INTERRUPT}$ du microprocesseur ; nous laissons le soin au lecteur d'imaginer quelle est alors la structure du programme à adopter.

III. 3. Programme pour incrémenter le nombre affiché

Le programme principal de la figure 12 nous montre qu'il faut incrémenter le nombre décimal de 4 chiffres affiché, à chaque fois qu'un front descendant sur l'entrée \overline{EF}_1 est détecté. Ceci est réalisé en modifiant correctement le contenu des 4 lignes mémoire constituant l'espace des données ; rappelons que la mémoire contient le caractère \square_4 , c'est-à-dire les unités, à l'adresse $n + 03$, \square_3 c'est-à-dire les dizaines à l'adresse $n + 02$, \square_2 c'est-à-dire les centaines à l'adresse $n + 01$, \square_1 c'est-à-dire les milliers à l'adresse n . La procédure à adopter est la suivante :

- a) Comparer le caractère \square_4 à 9 :
 - si $\square_4 \neq 9$, remplacer \square_4 par $\square_4 + 1$ et aller en e).
 - si $\square_4 = 9$, remplacer \square_4 par 0 et aller en b).
- b) Comparer le caractère \square_3 à 9 :
 - si $\square_3 \neq 9$, remplacer \square_3 par $\square_3 + 1$ et aller en e).
 - si $\square_3 = 9$, remplacer \square_3 par 0 et aller en c).
- c) Comparer le caractère \square_2 à 9 :
 - si $\square_2 \neq 9$, remplacer \square_2 par $\square_2 + 1$ et aller en e).
 - si $\square_2 = 9$, remplacer \square_2 par 0 et aller en d).
- d) Comparer le caractère \square_1 à 9 :
 - si $\square_1 \neq 9$, remplacer \square_1 par $\square_1 + 1$ et aller en e).
 - si $\square_1 = 9$, remplacer \square_1 par 0 et aller en e).
- e) Le nombre décimal de 4 chiffres est incrémenté.

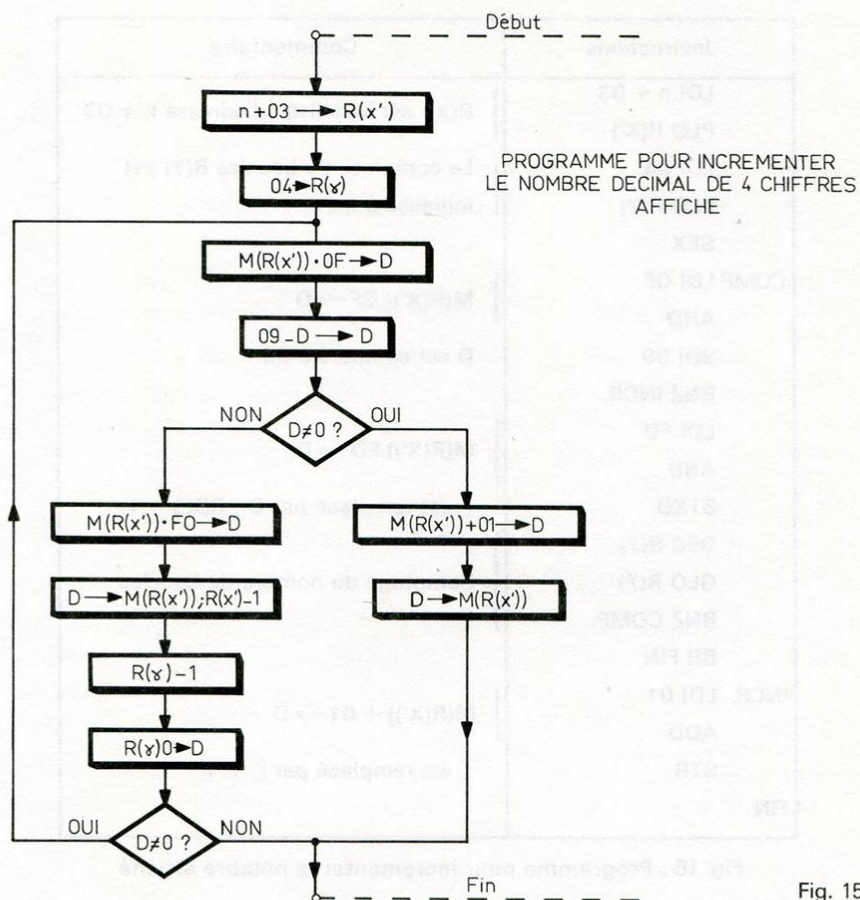


Fig. 15

Les 4 parties a) b) c) et d) sont identiques à l'indice du caractère près : le programme est répétitif et d'ailleurs aisément généralisable à un nombre quelconque de caractères. D'une façon générale, cette particularité est à rechercher quel que soit le problème à résoudre car elle permet d'adopter une structure de programme en boucle fermée qui consomme peu de lignes mémoire. L'organigramme est représenté figure 15 ; c'est $R(\alpha)$ qui est compteur de boucles et $R(X')$ pointeur des données (remarquez que dans notre exemple précis, on peut prendre respectivement $R(\alpha)$ et $R(X)$: $R(X')$ est initialement chargé par l'adresse $n + 03$ du caractère \square_4 , et à chaque fois que le programme décrit la boucle, une opération de stockage STXD est effectuée ce qui permet de descendre automatiquement l'indice du caractère d'une unité.

Instructions	Commentaire
LDI n + 03	} R(X') est initialisé à l'adresse n + 03
PLO R(X')	
LDI 04	} Le compteur de boucles R(?) est initialisé à 04
PLO R(?)	
SEX	
COMP. LDI 0F	} M(R(X')).0F → D
AND	
SDI 09	D est comparé à 09
BNZ INCR.	
LDI FO	} M(R(X')).FO → D
AND	
STXD	[est remplacé par 0 ; R(X') - 1
DEC R(?)	} Comptage du nombre de boucles
GLO R(?)	
BNZ COMP.	
BR FIN	
INCR. LDI 01	} M(R(X')) + 01 → D
ADD	
STR	[est remplacé par [+ 1
FIN	

Fig. 16 : Programme pour incrémenter le nombre affiché

Pour terminer l'examen de cet organigramme, signalons une petite difficulté liée au fait qu'il ne faut en aucun cas modifier les 4 bits de poids fort de chaque ligne de l'espace des données ; ceci est réalisé de la façon suivante :

- M étant l'octet contenu dans la ligne mémoire correspondant au caractère [, pour comparer [à 9 il faut tout d'abord effectuer un ET logique entre M et 0F ; le résultat de cette opération qui est contenu dans l'accumulateur D est ensuite comparé à l'octet 09.

- Pour remplacer [par 0 sans affecter les 4 bits de poids fort de M, il suffit d'effectuer un ET logique entre M et l'octet FO.

- Pour remplacer [par [+ 1 sans affecter les 4 bits de poids fort de M, il suffit d'additionner l'octet 01 à M.

La figure 16 représente en langage assembleur le programme d'incrémement correspondant à l'organigramme de la figure 15.

III. 4. Le programme complet du compteur

A titre d'exemple, nous donnons un programme complet correspondant à l'organigramme de la figure 12. Ce programme est donné en langage machine figure 17 et est constitué par la réunion du sous-programme d'affichage, du sous-programme de temporisation et du programme d'incrémement. 7 registres sont utilisés (hormis R (0) qui est compteur de programme principal) :

- R (3) est compteur de programmè pour le S.P. d'affichage.
- R (4) est compteur de programme pour le S.P. de temporisation.
- R (5) est le pointeur R (X) pour l'affichage.
- R (6) est le pointeur R (X') pour le programme d'incrémement.
- R (7), R (8), R (9) sont les compteurs de boucles respectivement R (α), R (β), R (γ).

Par ailleurs, nous avons placé l'espace des données aux adresses de 7 C à 7 F.

Le programme commence par une procédure d'initialisation qui consiste à :

- initialiser les compteurs R (3) et R (4) aux adresses respectivement des S.P. d'affichage et de temporisation.
- remplir l'espace des données par les octets 80, 40, 20, 10 de telle sorte que le premier nombre affiché soit 0000.

Ce programme est prévu pour fonctionner à une fréquence d'horloge de 1 MHz à cause de l'existence du sous-programme de temporisation ; on peut cependant choisir une fréquence différente à condition de modifier l'octet N écrit à l'adresse 49, octet qui définit la durée de la temporisation.

Pour terminer ce paragraphe, signalons que l'exemple que nous avons développé n'est pas limitatif : d'autres applications du même ordre sont évidemment envisageables. On peut par exemple imaginer le problème suivant :

- un distributeur automatique de tickets peut recevoir des pièces de monnaie de 2 F, 1 F, 50 c. et 20 c. Les pièces placées dans l'appareil coupent 4 faisceaux lumineux susceptibles de déclencher les entrées \overline{EF}_1 , \overline{EF}_2 , \overline{EF}_3 , \overline{EF}_4 du microprocesseur.
- le programme doit être tel qu'apparaisse à chaque instant sur les afficheurs le total de la somme versée.
- lorsque le prix du ticket est atteint, la sortie Q du microprocesseur doit passer au niveau 1 (ce qui simule la remise du ticket).

Nous laissons le soin au lecteur de résoudre ce problème.

<u>Initialisation</u>		<u>Le faisceau est-il toujours coupé ?</u>	
Adresse	Instructions		
00	F8 38 ENT.AFF	32 FIN	D3
02	A3	33	3C 13 TEST
03	F8 48 ENT. TEMP.	35	30 32 FIN
05	A4	<u>Sous programme d'affichage</u>	
06	F8 7F	37 RET.AFF.	D0
08	A5	38 ENT.AFF.	F8 7C
09	E5	3A	A5
0A	F8 10	3B	F8 04
0C	73	3D	A7
0D	FE	3E	E5
0E	73	3F OUTPUT	61
0F	FE	40	D4
10	73	41	27
11	FE	42	87
12	73	43	3A 3F OUTPUT
		45	30 37 RET.AFF.
<u>Test du drapeau EF₁</u>		<u>Sous-programme de temporisation</u>	
13 TEST	D3	47 RET.TEMP	D3
14	3C 13 TEST	48 ENT.TEMP	F8 14
<u>Incrémentation</u>		4A	A8
16	F8 7F	4B DECR.	28
18	A6	4C	88
19	F8 04	4D	3A 4B DECR.
1B	A9	4F	30 47 RET.TEMP
1C	E6		
1D COMP	F8 0F		
1F	F2		
20	FD 09		
22	3A 2E INCR.		
24	F8 F0		
26	F2		
27	73		
28	29		
29	89		
2A	3A 1D COMP		
2C	30 32 FIN		
2E INCR	F8 01		
30	F4		
31	56		

IV. Application à la chronométrie

IV. 1. But proposé

Nous proposons de montrer qu'une légère modification de la structure du programme précédent permet de réaliser un chronomètre. Nous désirons que ce chronomètre mesure, au centième de seconde près, l'intervalle de temps entre deux événements. Ces deux événements sont, d'une part la mise au niveau 0 de \overline{EF}_1 (qui déclenche le chronomètre) et d'autre part la mise au niveau 0 de \overline{EF}_2 (qui arrête le chronomètre). Par ailleurs, dès que le chronomètre est déclenché, nous décidons de mettre la sortie Q à 1 ce qui permet éventuellement de commander un organe électronique ; le caractère \square_4 affiché doit indiquer les centièmes de seconde, le caractère \square_3 les dixièmes, le caractère \square_2 les unités et le caractère \square_1 les dizaines. Le chronomètre étant arrêté, la sortie Q doit être remise à 0 et l'intervalle de temps écoulé présent sur les afficheurs. Pour finir, nous utilisons l'entrée \overline{EF}_3 pour réinitialiser le chronomètre :

- si $\overline{EF}_3 = 1$, un seul déclenchement est autorisé.
- si $\overline{EF}_3 = 0$, le système affiche 0000 et un déclenchement suivant est autorisé.

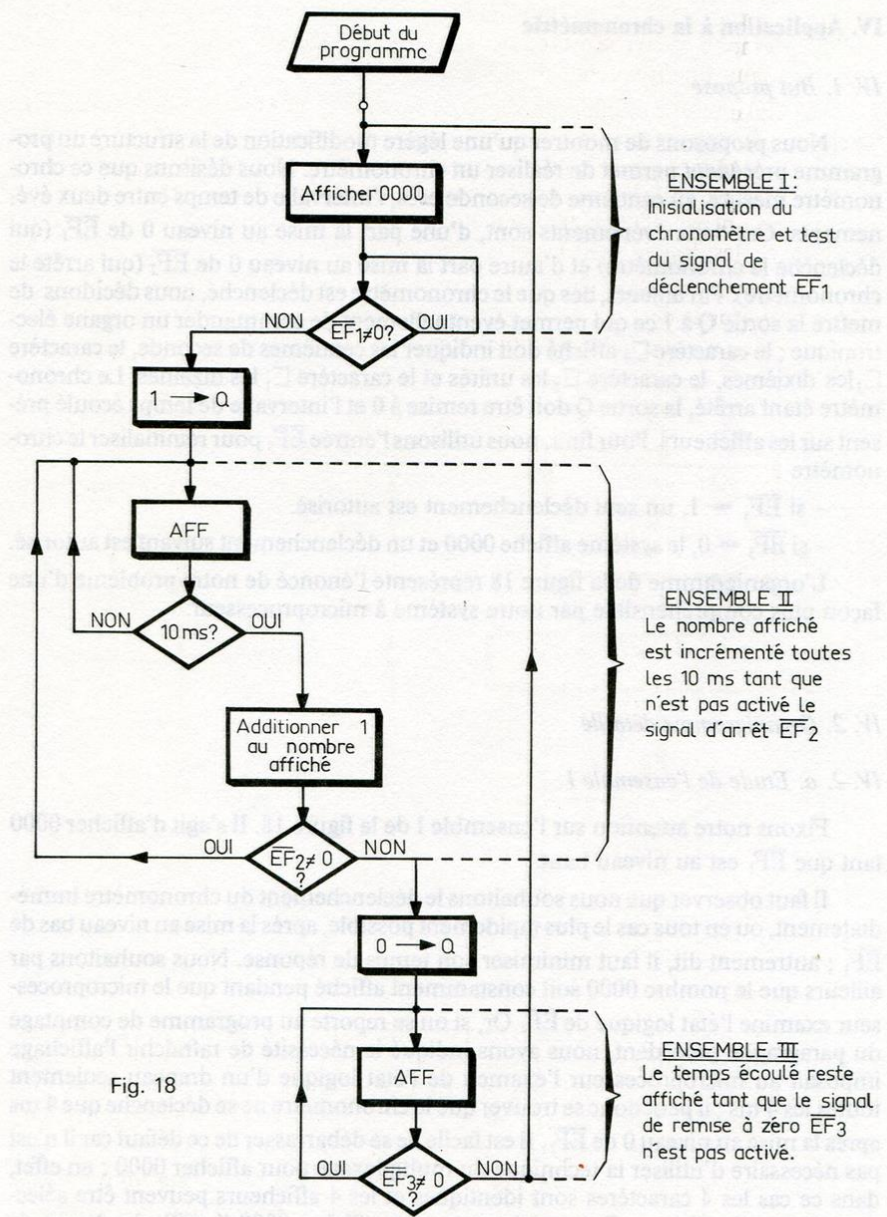
L'organigramme de la figure 18 représente l'énoncé de notre problème d'une façon plus compréhensible par notre système à microprocesseur.

IV. 2. Organigramme détaillé

IV. 2. a. Etude de l'ensemble I

Fixons notre attention sur l'ensemble I de la figure 18. Il s'agit d'afficher 0000 tant que \overline{EF}_1 est au niveau haut.

Il faut observer que nous souhaitons le déclenchement du chronomètre immédiatement, ou en tous cas le plus rapidement possible, après la mise au niveau bas de \overline{EF}_1 ; autrement dit, il faut minimiser son temps de réponse. Nous souhaitons par ailleurs que le nombre 0000 soit constamment affiché pendant que le microprocesseur examine l'état logique de \overline{EF}_1 . Or, si on se reporte au programme de comptage du paragraphe précédent, nous avons indiqué la nécessité de rafraîchir l'affichage imposait au microprocesseur l'examen de l'état logique d'un drapeau seulement toutes les 4 ms ; il peut donc se trouver que le chronomètre ne se déclenche que 4 ms après la mise au niveau 0 de \overline{EF}_1 . Il est facile de se débarrasser de ce défaut car il n'est pas nécessaire d'utiliser la technique du multiplexage pour afficher 0000 ; en effet, dans ce cas les 4 caractères sont identiques et les 4 afficheurs peuvent être sélectionnés simultanément. Par conséquent, pour afficher 0000 il suffit de charger le



registre de sortie CDP 1852 par l'octet FO ; aucun rafraîchissement n'est alors nécessaire et le temps de réponse est réduit dans le pire des cas à 2 cycles machine, c'est-à-dire 16 μ s pour une fréquence d'horloge de 1 MHz.

Cependant, le multiplexage s'impose lorsque le nombre à afficher peut être constitué par des caractères différents, ce qui est le cas lorsque le chronomètre tourne. On se rappelle que les codes des caractères sont alors contenus dans l'espace des données à partir de l'adresse n. Il s'ensuit que le chronomètre comptera le temps en commençant par afficher 0000 seulement s'il est prévu, dans l'ensemble I, de remplir l'espace des données par les octets 80, 40, 20, 10.

L'organigramme détaillé de l'ensemble I est donné figure 19. Remarquez que FO est une donnée destinée à être sortie vers les afficheurs, et l'organigramme de la figure 19 suppose que FO est écrit à l'adresse $n - 01$ de la mémoire.

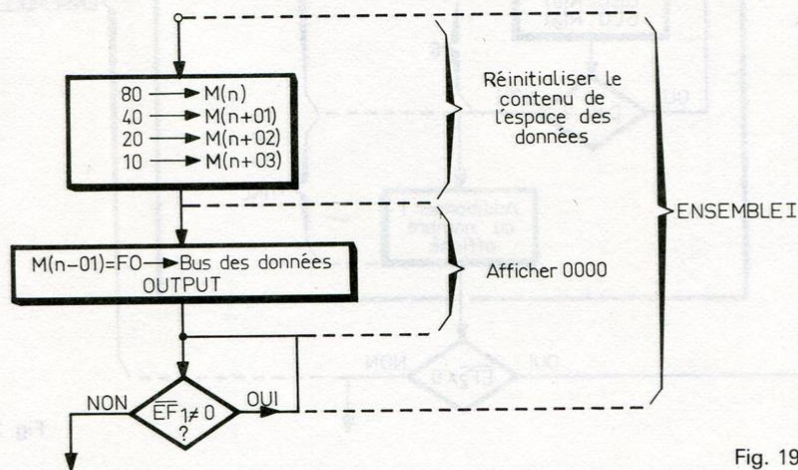


Fig. 19

IV. 2. b. Etude de l'ensemble II

L'ensemble II constitue le noyau central de notre chronomètre : il s'agit d'incrémenter le nombre affiché exactement toutes les 10 ms. Il est donc a priori nécessaire de faire plusieurs fois appel au sous-programme d'affichage ; ceci est réalisé en utilisant le compteur de boucles R (δ) : le nombre décimal N' chargé initialement dans R (δ) définit le nombre d'appels à effectuer (fig. 20).

L'ensemble II est constitué par le sous-ensemble II bouclé sur lui-même par l'intermédiaire du test du drapeau \overline{EF}_2 . Pour pouvoir régler la durée d'exécution du

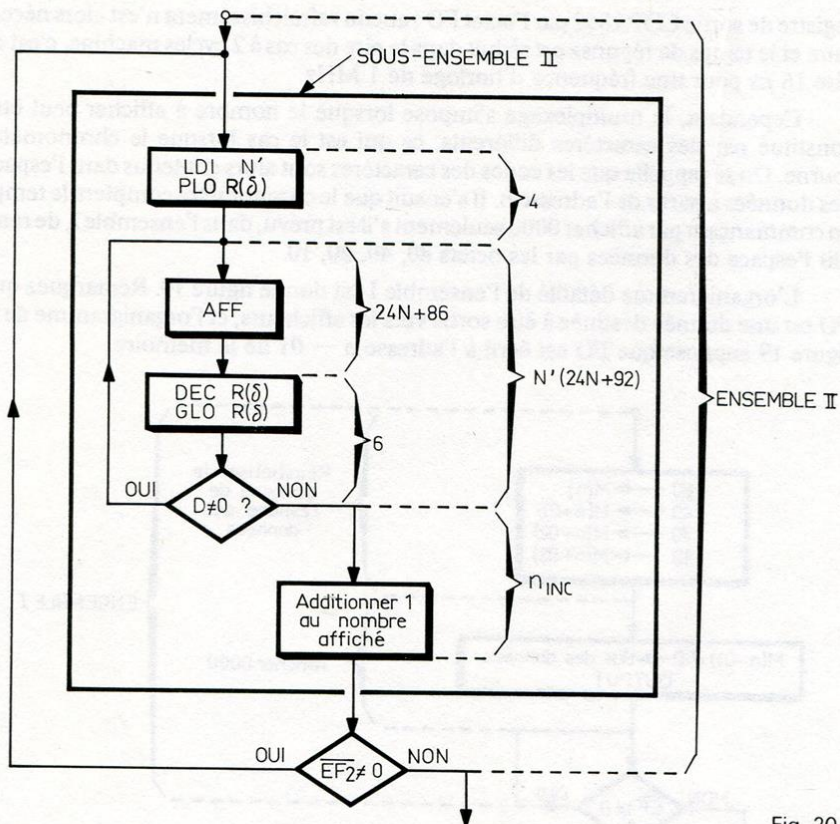


Fig. 20

sous-ensemble II à 10 ms, il est nécessaire de faire le décompte exact du nombre de cycles requis ; en particulier :

– le lecteur ayant fait l'effort d'examiner attentivement les figures 13 et 14 pourra constater que le nombre de cycles requis pour l'exécution du sous-programme d'affichage est $24N + 86$. N représente, en décimal, l'octet qui définit la durée du sous-programme de temporisation.

– le nombre de cycles n_{INC} requis pour l'exécution de l'incréméntation du nombre affiché n'est malheureusement pas constant. En effet, si l'on se reporte à la figure 15, on peut constater que la direction prise par ce programme dépend de l'existence ou non d'une retenue lors de l'addition à 01 :

- a) s'il n'y a pas de retenue, $n_{INC} = 24$.
- b) s'il y a une retenue sur le caractère \square_3 , $n_{INC} = 32$.

Adresse	Instructions	Mnémonique	Adresse	Instructions	Mnémonique
00	F8 4A	LDI ENT.AFF.	36	73	STXD
02	A3	PLO R(3)	37	29	DEC R(?)
03	F8 5A	LDI ENT.TEMP	38	89	GLO R(?)
05	A4	PLO R(4)	39	3A 2C COMP	BNZ
06	F8 7B	LDI n - 01	3B	30 41 FIN	BR
08	A5	PLO R(X)	3D INCR.	F8 01	LDI 01
09	F8 F0	LDI F0	3F	F4	ADD M (R(X'))
0B	55	STR M (R(X))	40	56	STR M (R(X'))
			41 FIN	3D 1D MIMI	BN 2
	<u>Ensemble I</u>			<u>Ensemble III</u>	
0C DEB	F8 7F	LDI n + 03	43	7A	REQ
0E	A5	PLO R (X)	44 TOTO	D3	SEP
0F	E5	SEX	45	3E 44 TOTO	BN 3
10	F8 10	LDI 10	47	30 0C DEB.	BR
12	73	STXD		<u>Sous-programme d'affichage</u>	
13	FE	SHL	49 RET.AFF	D0	SEP
14	73	STXD	4A ENT.AFF	F8 7C	LDI n
15	FE	SHL	4C	A5	PLO R(X)
16	73	STXD	4D	F8 04	LDI 04
17	FE	SHL	4F	A7	PLO R(x)
18	73	STXD	50	E5	SEX
19	61	OUT M (R(X))	51 OUTP.	61	OUT M (R(X))
1A DECL.	3C 1A DECL.	BN1	52	D4	SEP
	<u>Ensemble II</u>		53	27	DEC R(x)
1C	7B	SEQ	54	87	GLO R(x)
1D MIMI	F8 03	LDI N'	55	3A 51 OUTP.	BNZ
1F	AA	PLO R(δ)	57	30 49 RET.AFF.	BR
20 DEDE	D3	SEP		<u>Sous-programme de temporisation</u>	
21	2A	DEC R(δ)	59 RET.TEM.	D3	SEP
22	8A	GLO R(δ)	5A ENT.TEM.	F8 0D	LDI (13) déc.
23	3A 20 DEDE	BNZ	5C	A8	PLO R(β)
25	F8 7F	LDI n + 03	5D DECR.	28	DEC R(β)
27	A6	PLO R(X')	5E	88	GLO R(β)
28	F8 04	LDI 04	5F	3A 5D DECR.	BNZ
2A	A9	PLO R(?)	61	30 59 RET.TEM.	BR
2B	E6	SEX			
2C COMP	F8 0F	LDI 0F			
2E	F2	AND M (R(X'))			
2F	FD 09	SDI 09			
31	3A 3D INCR.	BNZ			
33	F8 F0	LDI F0			
35	F2	AND M(R(X'))			

Fig. 21

c) si la retenue est reportée sur le caractère \square_2 , $n_{INC} = 52$.

d) si la retenue est reportée sur le caractère \square_1 , $n_{INC} = 72$.

Le chronomètre aurait la précision du quartz utilisé pour faire fonctionner le microprocesseur, si l'égalité suivante était constamment respectée :

$$6 + n_{INC} + N' (24 N + 92) = \frac{10 \text{ ms}}{8T} = 1250 \text{ pour } T = 1 \mu s$$

En choisissant $N' = 3$ et $N = 13$, cette égalité devient : $n_{INC} + 1218 = 1250$.

La précision serait donc celle du quartz si n_{INC} étaient constamment égal à 32. Dans le meilleur des cas, les erreurs par excès dues aux retenues sont exactement compensées.

Notre chronomètre présente donc une erreur systématique (que nous estimons à environ 0,1 %) due à la structure du programme elle-même. Il est bien entendu envisageable d'imaginer un programme d'incrément du nombre affiché tel que sa durée d'exécution soit toujours la même, mais cela risque d'augmenter considérablement la taille mémoire ; par ailleurs le cerveau humain est mal utilisé puisqu'il est constamment occupé à faire le compte du nombre de cycles requis à l'exécution du programme qu'il désire mettre au point. Nous verrons au paragraphe suivant la façon de se libérer de ces contraintes.

Pour terminer, remarquez que le microprocesseur n'examine l'état logique de \overline{EF}_2 que toutes les 10 ms. Par conséquent à l'erreur relative systématique précédente de 0,1 % vient s'ajouter une erreur absolue égale au maximum à 10 ms quel que soit l'intervalle de temps mesuré.

Pour que le programme dont l'organigramme est donné figure 18 puisse se dérouler correctement, n'oublions pas qu'il est nécessaire de le faire précéder des instructions utiles à l'initialisation des compteurs de programme, des sous-programmes d'affichage et de temporisation, ainsi que de l'ordre de stockage de l'octet FO (pour afficher 0000) dans la mémoire à l'adresse $n - 01$.

Le programme est donné figure 21. Par rapport au programme de comptage du paragraphe précédent, nous avons utilisé le registre supplémentaire $R(A)$: $R(A)$ est le compteur de boucles $R(\delta)$.

V. Interruption de programme et précision :

V. 1. Principe

L'étude du chronomètre précédent nous a permis de nous rendre compte des contraintes imposées par le respect rigoureux d'une durée imposée d'exécution d'un programme. Notre problème était d'ajouter 1 au nombre affiché, exactement toutes les 10 ms ; ceci peut être réalisé à condition d'accepter une légère augmenta-

tion de la complexité de notre montage. En effet, supposez que l'opération « additionner 1 au nombre affiché » constitue le sous-programme d'interruption de notre chronomètre : il est clair que cette opération sera effectuée exactement toutes les 10 ms à condition d'appeler ce sous-programme exactement toutes les 10 ms.

Pour réaliser notre projet il faut donc disposer d'un générateur capable d'activer l'entrée INTERRUPT du microprocesseur exactement toutes les 10 ms.

V. 2. Le générateur d'interruptions

La figure 22 indique comment on peut réaliser un tel générateur. Le signal d'interruption est fourni par une bascule JK déclenchée par la sortie Q_n d'un diviseur par 2^n (fig. 23) ; dès que la requête d'interruption est acquiescée, le signal SC1 remet la sortie \bar{Q} de la bascule JK au niveau 1. Remarquez que cette bascule est câblée de la même façon que sur la maquette A et remplit la même fonction.

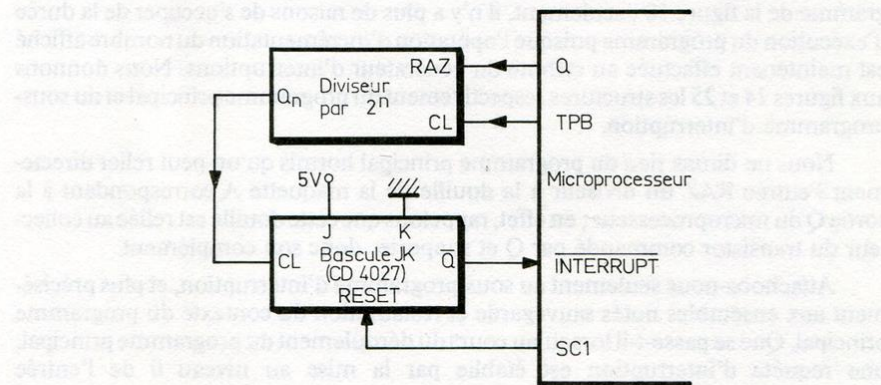


Fig. 22

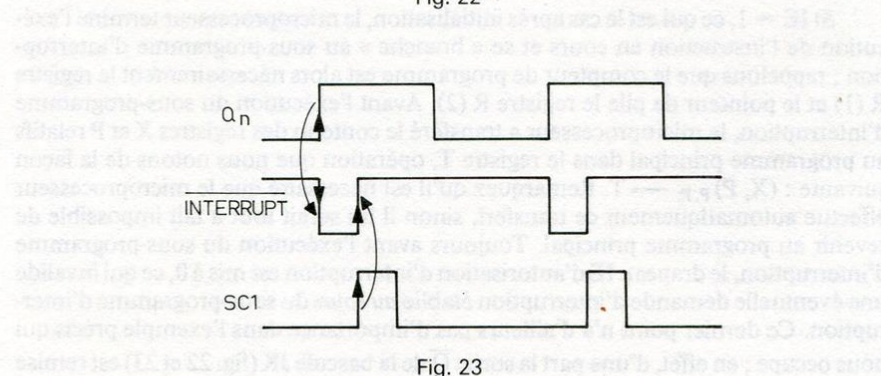


Fig. 23

Le diviseur par 2^n est tout simplement un compteur à n étages, Q_n étant la sortie du dernier étage ; le signal TPB dont la fréquence est 8 fois plus petite que la fréquence F de l'horloge du microprocesseur, déclenche le premier étage du diviseur. Notons que l'entrée de remise à zéro du compteur RAZ peut être commandée par la sortie Q du microprocesseur.

Il apparaît que la fréquence des signaux d'interruption est égale à $\frac{F}{8 \times 2^n}$. Par conséquent, si $n = 11$, il est nécessaire d'utiliser un quartz de fréquence $F = 1638,4$ kHz pour être en mesure d'activer l'entrée INTERRUPT exactement toutes les 10 ms. Dans la pratique, on pourra choisir le circuit CD 4040 comme diviseur, avec $Q_n = Q_{11}$.

V. 3. Mise en œuvre de l'interruption

Pour réaliser un chronomètre de précision, nous pouvons reprendre l'organigramme de la figure 18 ; seulement, il n'y a plus de raisons de s'occuper de la durée d'exécution du programme puisque l'opération d'incrément du nombre affiché est maintenant effectuée au rythme du générateur d'interruptions. Nous donnons aux figures 24 et 25 les structures respectivement du programme principal et du sous-programme d'interruption.

Nous ne dirons rien du programme principal hormis qu'on peut relier directement l'entrée RAZ du diviseur à la douille de la maquette A correspondant à la sortie Q du microprocesseur ; en effet, rappelons que cette douille est reliée au collecteur du transistor commandé par Q et supporte donc son complément.

Attachons-nous seulement au sous-programme d'interruption, et plus précisément aux ensembles notés sauvegarde et restauration du contexte du programme principal. Que se passe-t-il lorsqu'au cours du déroulement du programme principal, une requête d'interruption est établie par la mise au niveau 0 de l'entrée INTERRUPT ?

Si $IE = 1$, ce qui est le cas après initialisation, le microprocesseur termine l'exécution de l'instruction en cours et se « branche » au sous-programme d'interruption ; rappelons que le compteur de programme est alors nécessairement le registre $R(1)$ et le pointeur de pile le registre $R(2)$. Avant l'exécution du sous-programme d'interruption, le microprocesseur a transféré le contenu des registres X et P relatifs au programme principal dans le registre T , opération que nous notons de la façon suivante : $(X, P)_{P.P.} \rightarrow T$. Remarquez qu'il est nécessaire que le microprocesseur effectue automatiquement ce transfert, sinon il lui serait tout à fait impossible de revenir au programme principal. Toujours avant l'exécution du sous-programme d'interruption, le drapeau IE d'autorisation d'interruption est mis à 0, ce qui invalide une éventuelle demande d'interruption établie *au cours* du sous-programme d'interruption. Ce dernier point n'a d'ailleurs pas d'importance dans l'exemple précis qui nous occupe ; en effet, d'une part la sortie \overline{Q} de la bascule JK (fig. 22 et 23) est remise

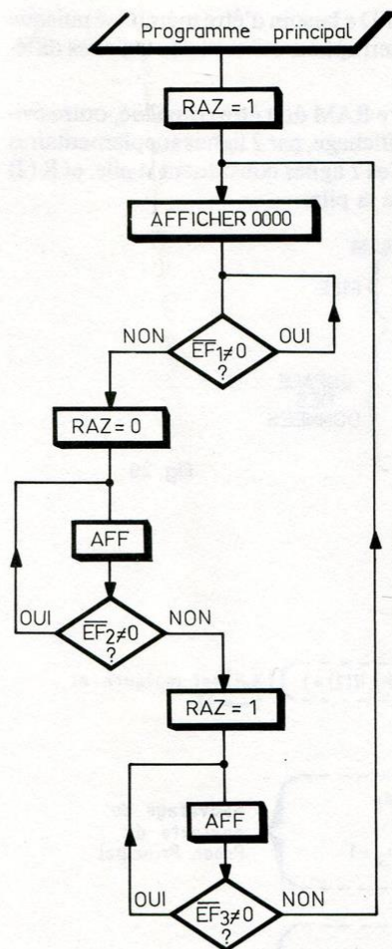


Fig. 24

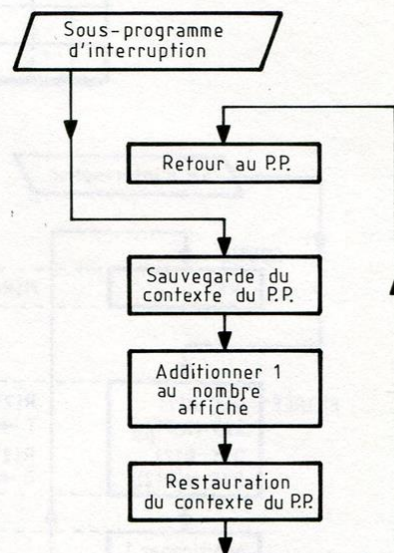


Fig. 25

au niveau 1 dès que l'interruption est accordée, d'autre part il est implicitement supposé que la durée totale d'exécution du sous-programme d'interruption est inférieure à 10 ms.

On appelle « contexte » d'un programme, le contenu de tous les registres utilisés au cours du déroulement de ce programme. Le sous-programme d'interruption peut utiliser ces mêmes registres à condition que, sa tâche étant terminée, il soit capable de restituer leur contenu au programme principal. Dans l'exemple que nous

développons, seul le contenu de l'accumulateur D a besoin d'être mémorisé puisque programme principal et sous-programme d'interruption utilisent des registres différents hormis précisément l'accumulateur D.

La figure 26 indique que l'espace mémoire RAM doit être constitué, outre évidemment l'espace des données nécessaire à l'affichage, par 2 lignes supplémentaires destinées à sauvegarder $(X, P)_{p.p.}$ et $(D)_{p.p.}$. Ces 2 lignes constituent la pile, et R(2) est pointeur de pile, n_p est l'adresse du bas de la pile.

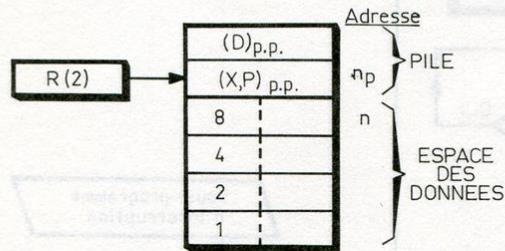


Fig. 26

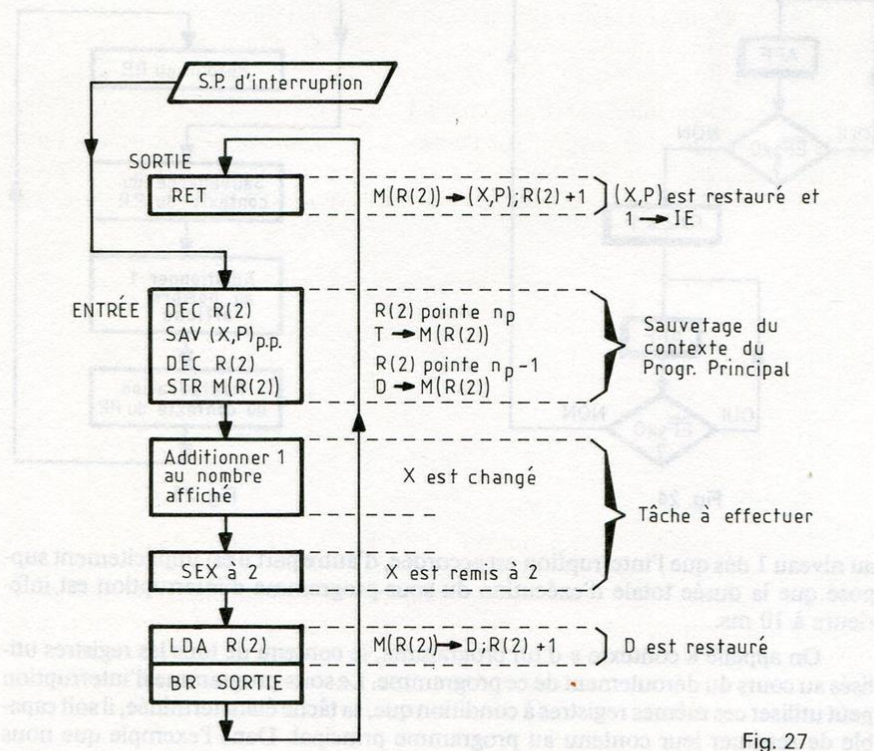


Fig. 27

La figure 27 indique de façon explicite la structure du sous-programme d'interruption. Pour que celui-ci puisse se dérouler normalement, il est nécessaire que R (2) soit initialisé à $n_p + 1$; ceci doit être fait au début du programme principal. Le lecteur qui désirera développer entièrement cet exemple n'oubliera pas non plus d'initialiser le compteur de sous-programme R (1).

VI. Application à la mesure des vitesses de rotation

VI. 1. Principe

Nous allons montrer comment le programme de comptage dont l'organigramme est donné figure 12 peut être transformé en un programme exécutant la mesure et l'affichage de la vitesse de rotation d'un moteur.

Pour capter une vitesse de rotation, plusieurs procédés sont envisageables. On peut par exemple imaginer une pale solidaire de l'arbre, qui coupe un faisceau lumineux à chaque tour ; ou bien, ce qui est plus pratique, un aimant fixé sur l'arbre peut induire aux bornes d'un bobinage relié à l'entrée d'un comparateur, une force électromotrice de fréquence égale à la fréquence de rotation du moteur.

En tout état de cause, le signal de période Δt qui représente la vitesse de rotation du moteur et qui est à envoyer sur l'entrée $\overline{EF_1}$ du microprocesseur, a la forme donnée figure 28. Si nous utilisons tel quel le programme du compteur donné au paragraphe III. 4., que va-t-il se passer dans le cas par exemple d'une vitesse de rotation de 100 tours par seconde ?

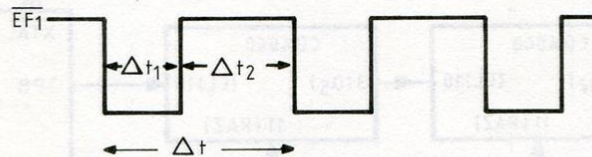


Fig. 28

Eh bien vous verrez tout simplement affiché le nombre de tours effectués, c'est-à-dire que vous verrez le nombre affiché constamment défiler, ce nombre s'incrémentant tous les centièmes de seconde. Supposez maintenant qu'on modifie le programme du compteur de telle sorte qu'il n'affiche pas le nombre incrémenté. Ce nombre, incrémenté à chaque tour et qui ne doit plus être affiché, doit cependant être présent dans 4 lignes de mémoire RAM que nous appellerons « espace de calcul ». Quel est le résultat de cette modification dans le cas par exemple d'une vitesse de rotation 100 tours par seconde ?

Comme le contenu des 4 lignes mémoire correspondant au nombre affiché et que nous appellerons « espace d'affichage » n'est pas affecté, vous verrez constamment affiché le nombre 0000. N'oubliez cependant pas que le nombre contenu dans l'espace de calcul s'incrémente ici tous les centièmes de seconde. Supposez pour terminer que, exactement toutes les secondes, on interrompe le cours du déroulement de notre programme modifié, en vue de demander au microprocesseur d'effectuer les opérations suivantes :

- transférer le contenu de l'espace de calcul dans l'espace d'affichage.
- remettre à zéro le contenu de l'espace de calcul.

C'est simple : il est affiché le nombre de tours effectués par seconde, c'est-à-dire la vitesse ou si on préfère, la fréquence.

La mise en œuvre de cette application nécessite donc la réalisation d'un générateur d'interruptions réglé à 1 Hz. Nous donnons figure 29 le schéma d'un tel générateur utilisant deux compteurs CD 4040 et une bascule JK CD 4027 ; les deux compteurs constituant un diviseur par 2^{17} , on fournit une interruption par seconde à condition d'utiliser un quartz de fréquence 1048,576 kHz. La sortie Q du microprocesseur peut être utilisée à commander l'entrée RAZ du diviseur ; rappelons que si on utilise la maquette A, la douille correspondant à la sortie Q supporte le complément de Q. Dans ce cas c'est l'instruction REQ de code 7 A qui a pour effet de remettre à zéro le diviseur.

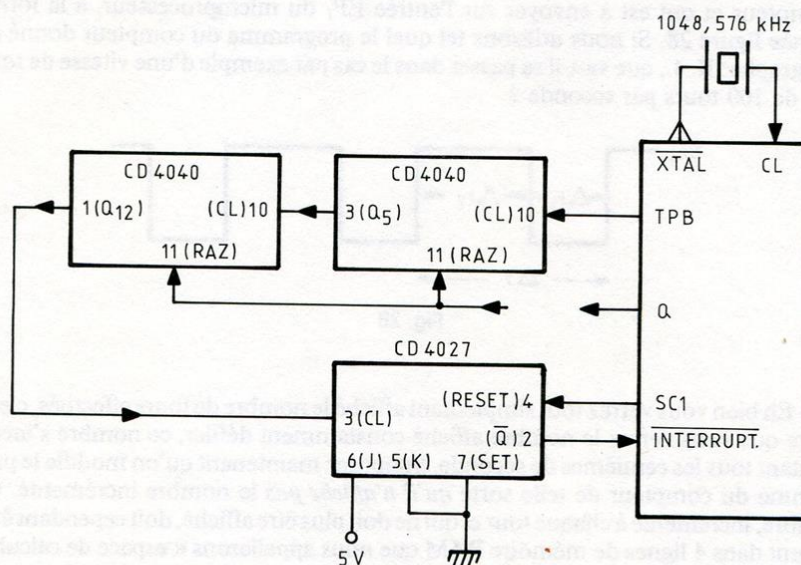


Fig. 29

Avant de rédiger le programme de ce tachymètre, il faut définir l'espace mémoire RAM réservé à l'espace de calcul, à l'espace d'affichage et à la pile. Cet espace mémoire peut être celui donné figure 30 ; il appelle deux remarques :

- Les deux lignes de la pile sont destinées à sauvegarder le contenu du registre T ce qui permet un retour au programme principal, ainsi que le contenu de l'accumulateur D.

- Le stockage des demi-octets 8, 4, 2, 1 dans l'espace de calcul et dans l'espace d'affichage doit être prévu tout au début du programme principal.

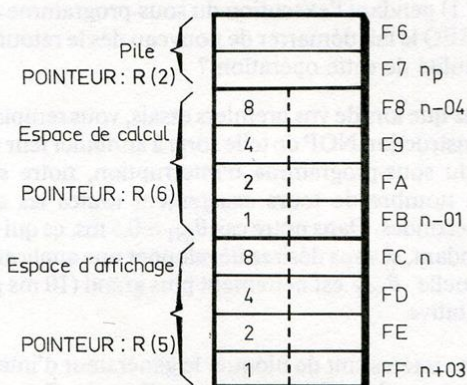


Fig. 30

VI. 2. Le sous-programme d'interruption

Le programme principal étant pratiquement identique au programme du compteur vu au début de ce chapitre, nous n'en dirons rien. Signalons seulement qu'il ne faut pas oublier d'initialiser les registres pointeurs et les registres compteurs de programme, ni de remplir les espaces de calcul et d'affichage par les demi-octets 8, 4, 2, 1.

En ce qui concerne le sous-programme d'interruption, nous avons déjà donné figure 27 la liste des instructions nécessaires à la bonne marche des opérations de sauvetage et de restauration du contexte. Il nous reste simplement à indiquer que pour le transfert du contenu de l'espace de calcul dans l'espace d'affichage, nous avons choisi respectivement les registres R (F) et R (E) comme pointeurs de ces espaces et adopté la procédure suivante :

- R (F) est initialisé à F 8, adresse du haut de l'espace de calcul ; R (E) est initialisé à F C, adresse du haut de l'espace d'affichage.

- M (R (F)) est transféré dans l'accumulateur D, et le contenu de D est stocké dans M (R (E)), puis R (F) et R (E) sont incrémentés. Cette opération est répétée 4 fois.

VI. 3. Le programme complet du tachymètre

Ce programme qui est donné figure 31 en langage machine, n'offre pas de difficultés particulières de déchiffage puisqu'il est constitué de sous-ensembles que nous avons préalablement étudiés. Revenons cependant un moment au sous-programme d'interruption dans lequel vous remarquerez que nous avons rajouté aux adresses 5 F et 81 respectivement les instructions REQ et SEQ. Si la douille de la maquette A correspondant à la sortie Q du microprocesseur est reliée à l'entrée RAZ du générateur d'interruptions, l'instruction REQ a pour effet de bloquer ce générateur ($RAZ = 1$) pendant l'exécution du sous-programme d'interruption, tandis que l'instruction SEQ le fait démarrer de nouveau dès le retour au programme principal. Quelle est l'utilité de cette opération ?

Supposez que lors de vos premiers essais, vous remplacez les instructions REQ et SEQ par l'instruction NOP de telle sorte à anihiler leur effet. Si θ_{INT} est la durée d'exécution du sous-programme d'interruption, notre système n'effectue pas le comptage du nombre de tours exactement toutes les secondes mais toutes les $(1 - \theta_{INT})$ secondes. Dans notre cas, $\theta_{INT} \approx 0,5$ ms, ce qui nous fait une erreur négligeable. Cependant, si vous désirez développer une application de plus grande envergure pour laquelle θ_{INT} est nettement plus grand (10 ms par exemple), cette erreur devient prohibitive.

Il est donc intéressant de bloquer le générateur d'interruptions pendant tout le temps que la mesure de la vitesse n'est pas effectuée. Cette amélioration a une conséquence importante que nous pourrions comprendre en examinant avec soin le fonctionnement du diviseur de la figure 29 : la sortie Q_{12} du compteur CD 4040, destinée à déclencher la bascule JK, est imposée au niveau 0 tant que $RAZ = 1$; à partir de l'instant où RAZ est mis à 0, il faut attendre seulement 0,5 s. avant que n'apparaisse un front montant en Q_{12} , et ceci bien que la fréquence du signal en Q_{12} soit égale à 1 Hz. Par conséquent, le simple fait d'introduire l'instruction REQ à l'adresse 5 F de la mémoire, a pour effet de générer une interruption toutes les demi-secondes et non plus toutes les secondes. Le nombre affiché est donc égal exactement à la moitié de la vitesse de rotation du moteur. Ceci peut être récupéré en réglant la période du générateur d'interruptions à 2 secondes, ou en demandant au programme principal d'incrémenter 2 fois par tours le nombre à afficher toutes les demi-secondes.

Le fréquencemètre à microprocesseur que nous venons de réaliser présente le défaut de ne pas être capable de mesurer des vitesses de rotation très élevées. Ceci est dû à la nécessité de rafraîchir périodiquement l'affichage. Pour se libérer de cette contrainte, on peut choisir l'une ou l'autre des solutions suivantes :

- réaliser un circuit d'interface entre le microprocesseur et les afficheurs en utilisant un décodeur par caractère. Ceci évite le multiplexage et donc le rafraîchissement, mais alourdit la structure matérielle.

<u>Initialisation</u>					
00	F8 5F ENT.INT.	3D	F4	6E	5E
02	A1	3E	56	6F	1E
03	F8 F8 np + 1	<u>Drapeau toujours mis ?</u>		70	4F
05	A2	3F FIN	D3	71	5E
06	F8 45 ENT. AFF.	40	3C 20 TEST	72	1E
08	A3	42	30 3F FIN	73	0F
09	F8 55 ENT.TEMP	<u>S.P. d'affichage</u>		74	5E
0B	A4	44 RET.AFF.	D0	75	EF
0C	F8 FF n + 03	45 ENT.AFF.	F8 FC n	76	F8 10
0E	A5	47	A5	78	73
0F	E5	48	F8 04	79	FE
10	F8 02	4A	A7	7A	73
12	A7	4B	E5	7B	FE
13 TOTO	27	4C OUT.	61	7C	73
14	F8 10	4D	D4	7D	FE
16	73	4E	27	7E	73
17	FE	4F	87	7F	E2
18	73	50	3A 4C OUT.	80	42 (D restauré)
19	FE	52	30 44 RET.AFF	81	7B (RAZ = 0)
1A	73	<u>S.P. temporisation</u>		82	30 5E
1B	FE	54 RET.TEMP.	D3		
1C	73	55 ENT.TEMP.	F8 02		
1D	87	57	A8		
1E	3A 13 TOTO	58 DECR.	28		
<u>Test du drapeau</u>		59	88		
20 TEST	D3	5A	3A 58 DECR.		
21	3C 20 TEST	5C	30 54 RET.TEMP		
<u>Incrémentation de l'espace de calcul</u>		<u>S.P. d'interruption</u>			
23	F8 FB n - 01	5E RET.INT.	70		
25	A6	5F ENT.INT.	7A (RAZ = 1)		
26	F8 04	60	22		
28	A9	61	78 (SAV T)		
29	E6	62	22		
2A COMP	F8 0F	63	52 (SAV D)		
2C	F2				
2D	FD 09	64	F8 F8 n + 04		
2F	3A 3B INCR	66	AF		
31	F8 F0	67	F8 FC n		
33	F2	69	AE		
34	73	6A	4F		
35	29	6B	5E		
36	89	6C	1E		
37	3A 2A COMP	6D	4F		
39	30 3F FIN				
3B INCR	F8 01				

Fig. 31

– accepter de ne pas afficher la vitesse pendant sa mesure mais seulement si un drapeau est mis (EF_2 par exemple). Nous conseillons au lecteur de rédiger le programme adéquat.

L'intérêt d'utiliser un microprocesseur pour mesurer une vitesse réside dans le fait qu'elle est présente dans la mémoire sous forme numérique. Elle peut donc être facilement « traitée » par le microprocesseur en vue, par exemple, du réglage précis et automatique de l'avance à l'allumage dans le cas de l'automobile. Cette vitesse numérisée peut aussi être constamment comparée à une vitesse « idéale » en vue d'obtenir sa régulation. Ou bien encore, on peut imaginer la possibilité de mesurer et d'afficher des variations de vitesse par simple calcul de différences.

VII. Application aux systèmes programmables

Le possesseur de l'ensemble maquette A, maquette B, ayant réalisé le circuit d'interfaçage de la figure 6 ainsi que le générateur d'interruptions, voit s'ouvrir devant lui un vaste champ d'applications. Nous ne tenterons pas, faute de place, de le défricher.

Cependant il nous paraît opportun de donner toutes les indications utiles à la réalisation de systèmes programmables. Qu'entendons-nous par là ?

VII. 1. Principe

Prenons un exemple concret : supposez que vous désiriez fabriquer un temporisateur susceptible de déclencher un engin au bout d'un temps θ . Pour que ce temporisateur soit commode d'emploi, il apparaît qu'il doit autoriser les deux modes de fonctionnement suivants :

– *mode « programmation »* : l'utilisateur est en mesure de présenter sur les afficheurs, c'est-à-dire d'entrer dans le système, le temps θ souhaité. Au cours de cette opération le temporisateur est bloqué. Nous dirons que le temps θ est programmable.

– *mode « décomptage du temps programmé »* : l'utilisateur n'a plus accès au système, et le temporisateur effectue la tâche qui lui incombe, c'est-à-dire le déclenchement de l'engin au bout du temps θ préalablement programmé.

Pour que la réalisation d'un tel système programmable soit envisageable, il faut prévoir la possibilité d'introduire les 4 caractères correspondant au temps θ demandé. Ceci se fait ordinairement au moyen d'un clavier, mais on peut tout aussi bien imaginer d'utiliser les 4 entrées \overline{EF} du microprocesseur. En effet, nous suggérons que dans le mode « programmation » du temporisateur, une action sur l'entrée \overline{EF}_i ait l'effet suivant :

- si $\overline{EF}_i = 1$, il ne se passe rien.
- si $\overline{EF}_i = 0$, le caractère \square_i et seulement celui-là subit une « rotation » environ toutes les demi-secondes.

Autrement dit, tant que $\overline{EF}_i = 0$, le caractère \square_i est incrémenté toutes les demi-secondes jusqu'à sa valeur maximale, puis repasse à 0 et ainsi de suite.

Ce procédé a le mérite d'éviter la lourdeur logicielle qui résulterait de la scrutation et du décodage d'un clavier.

Un cinquième drapeau nous est nécessaire pour sélectionner l'un des deux modes de fonctionnement de notre système. Nous pouvons décider qu'il s'agit de la clé D_7 de la maquette A, et qu'un niveau logique 1 sur cette clé impose le mode « programmation ». Pour tester l'état de ce drapeau supplémentaire, il faut demander au programme d'exécuter les instructions suivantes :

- mettre le mot présent sur les clés dans l'accumulateur D (INPUT), puis opérer un décalage à gauche (SHL) du contenu de D.
- le niveau imposé sur la clé D_7 se retrouve donc dans le registre DF, dont il suffit de tester l'état.

Il résulte de tout ceci que le programme principal de notre temporisateur ou de tout autre système programmable du même genre, doit présenter la structure générale de la figure 32. Tant que nous sommes dans le mode « programmation », le générateur d'interruptions est bloqué ($RAZ = 1$), et le système attend que vous lui

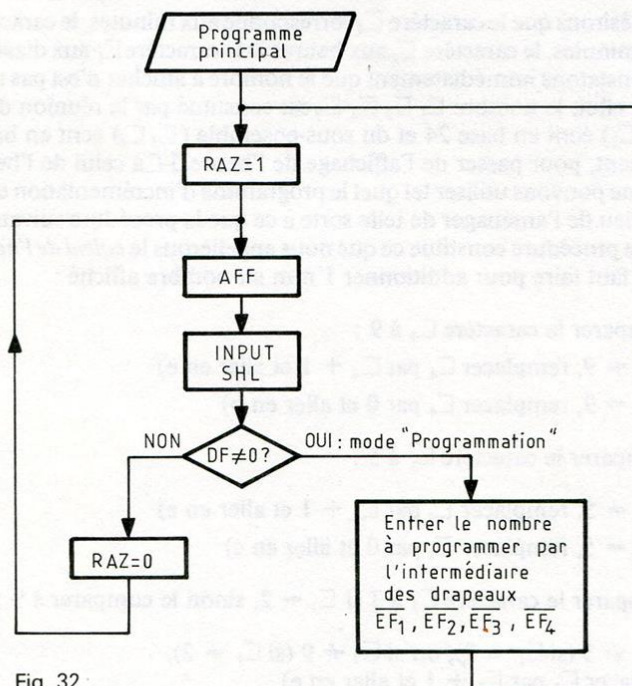


Fig. 32

fournissiez la donnée (le temps θ) nécessaire à l'exécution de la tâche qu'il a à accomplir (décompter le temps θ). La mise au niveau 0 de la clé D, a pour effet de débloquent le générateur d'interruptions ($RAZ = 0$) et par conséquent d'engager périodiquement le système dans le sous-programme d'interruption. Si ce sous-programme est appelé toutes les secondes et contient les instructions nécessaires à la décrémentation du nombre affiché, ainsi que les instructions relatives au déclenchement (par la sortie N_1 du microprocesseur par exemple) d'un engin lorsque ce nombre vaut zéro, vous avez réalisé un temporisateur programmable qui répond aux spécifications que nous avons préalablement annoncées.

VII. 2. Exemple d'application : horloge

Pour donner des indications plus détaillées concernant la mise en œuvre des systèmes programmables, nous allons demander au microprocesseur d'afficher l'heure exacte. Le système est programmable dans le sens où il doit permettre une mise à l'heure dans une première étape ; dans une seconde étape que nous ne développerons pas ici, il doit permettre de programmer les époques du déclenchement et de l'arrêt d'un certain nombre d'engins à commander automatiquement.

Nous désirons que le caractère \square_4 corresponde aux minutes, le caractère \square_3 aux dizaines de minutes, le caractère \square_2 aux heures et le caractère \square_1 aux dizaines d'heures. Nous constatons immédiatement que le nombre à afficher n'est pas un nombre décimal. En effet, le nombre $\square_1 \square_2 \square_3 \square_4$ est constitué par la réunion du sous-ensemble ($\square_1 \square_2$) écrit en base 24 et du sous-ensemble ($\square_3 \square_4$) écrit en base 60. Par conséquent, pour passer de l'affichage de l'heure $\square \square$ à celui de l'heure $\square \square + 1$ mm, nous ne pouvons utiliser tel quel le programme d'incrémenté donné figure 16. Il y a lieu de l'aménager de telle sorte à ce que la procédure suivante soit respectée ; cette procédure constitue ce que nous appellerons le *calcul de l'heure* et indique ce qu'il faut faire pour additionner 1 mm au nombre affiché :

- a) Comparer le caractère \square_4 à 9 :
 - si $\square_4 \neq 9$, remplacer \square_4 par $\square_4 + 1$ et aller en e)
 - si $\square_4 = 9$, remplacer \square_4 par 0 et aller en b)
- b) Comparer le caractère \square_3 à 5 :
 - si $\square_3 \neq 5$, remplacer \square_3 par $\square_3 + 1$ et aller en e)
 - si $\square_3 = 5$, remplacer \square_3 par 0 et aller en c)
- c) Comparer le caractère \square_2 à 3 si $\square_1 = 2$, sinon le comparer à 9 :
 - si $\square_2 \neq 3$ (si $\square_1 = 2$), ou si $\square_2 \neq 9$ (si $\square_1 \neq 2$), remplacer \square_2 par $\square_2 + 1$ et aller en e)

– si $\square_2 = 3$ (si $\square_1 = 2$), ou si $\square_2 = 9$ (si $\square_1 \neq 2$), remplacer \square_2 par 0 et aller en d)

d) Comparer le caractère \square_1 à 2.

– si $\square_1 \neq 2$, remplacer \square_1 par $\square_1 + 1$ et aller en e)

– si $\square_1 = 2$, remplacer \square_1 par 0 et aller en e)

e) C'est terminé : il a été rajouté 1 minute à l'heure affichée.

Cette procédure est fort semblable à celle utilisée pour incrémenter un nombre décimal, mais diffère en deux points :

– chaque caractère est à comparer avec un nombre différent

– le caractère \square_2 est à comparer avec un nombre qui dépend du caractère \square_1 .

Ainsi, pour conserver la même structure de programme (fig. 15), il apparaît comme nécessaire que les 4 opérandes – 9 pour le caractère \square_4 , 5 pour le caractère \square_3 , 3 ou 9 pour le caractère \square_2 , 2 pour le caractère \square_1 – soient rangés dans la mémoire dans une zone que nous appellerons « espace des opérandes ». Cet espace est représenté figure 33 et est pointé par le registre R (6) par exemple. Pour que l'incréméntation de l'heure puisse correctement s'effectuer, le programme de calcul de l'heure doit préalablement stocker dans la mémoire à l'adresse $n' + 2$, soit l'octet 43 soit l'octet 49, et ceci selon la valeur du caractère \square_1 . Ce caractère étant contenu dans l'espace d'affichage à l'adresse n, sa reconnaissance requiert l'utilisation d'un registre – R (A) – pointant constamment la ligne d'adresse n. Le registre R (5) est par ailleurs utilisé

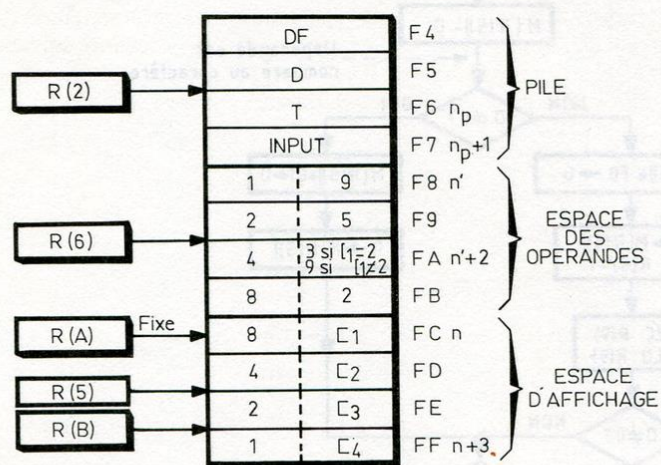


Fig. 33

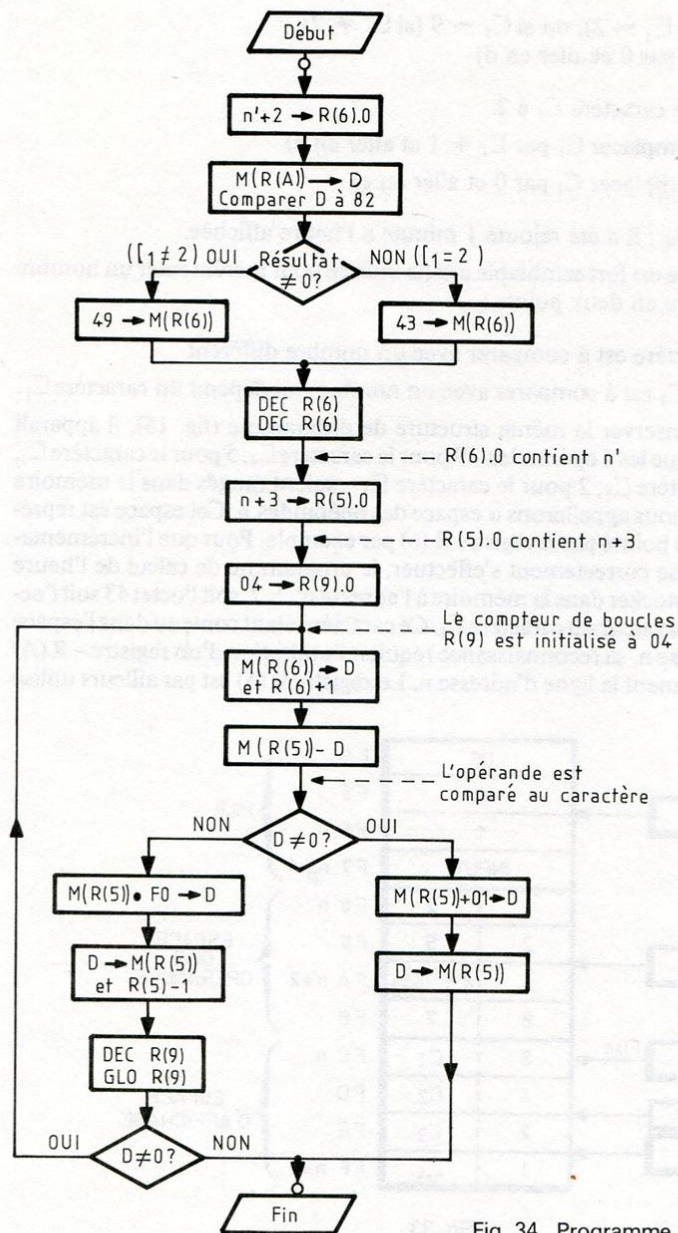


Fig. 34. Programme de calcul de l'heure

pour pointer une ligne quelconque de l'espace d'affichage. La figure 34 représente l'organigramme détaillé du programme de calcul de l'heure ; notez la similitude avec l'organigramme du programme d'incrément d'un nombre décimal de 4 chiffres donné figure 15.

Le calcul de l'heure doit être effectué exactement toutes les 60 secondes. Nous savons que pour obtenir la précision du quartz, il est commode de demander au sous-programme d'interruption l'exécution de cette tâche. Pour que nous puissions utiliser notre générateur précédent battant à la seconde, il est nécessaire que le calcul de l'heure ne s'opère effectivement que tous les 60 appels de sous-programme d'interruption. Pour cela, utilisons un compteur de boucles – R (F) par exemple – que nous supposons préalablement initialisé par le programme principal, à la valeur 60 en décimal, soit 3C en hexadécimal. L'organigramme du sous-programme d'interruption de la figure 35 indique clairement qu'il est ajouté une minute à l'heure

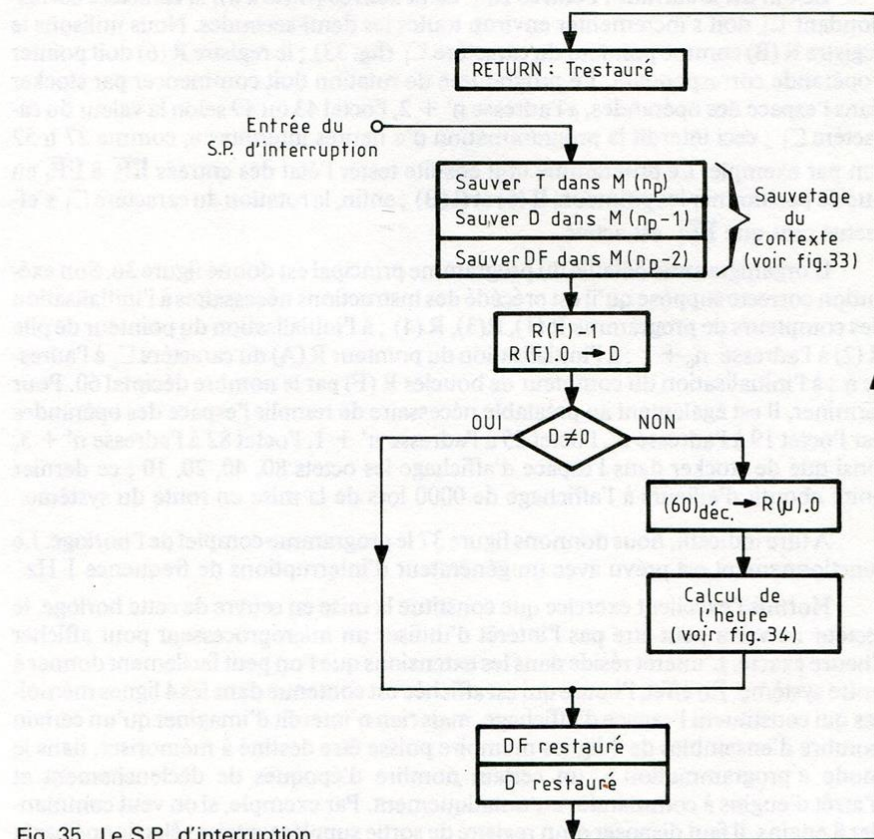


Fig. 35. Le S.P. d'interruption

affichée seulement toutes les 60 secondes, ce qui aboutit naturellement au calcul de l'heure exacte. Il faut par ailleurs remarquer que nous avons pris la peine de sauvegarder le drapeau DF, et ceci dans le seul but de prévoir une extension possible pour laquelle ce drapeau serait utilisé dans le sous-programme d'interruption.

Nous savons maintenant comment destiner le microprocesseur au calcul de l'heure, mais nous ne lui avons pas encore donné les instructions nécessaires à l'initialisation de notre horloge. Cette tâche incombe au programme principal et constitue le fonctionnement dans le mode « programmation ». Il s'agit de stocker l'heure initiale dans l'espace d'affichage par l'intermédiaire des entrées \overline{EF}_1 à \overline{EF}_4 du microprocesseur. Comme nous avons suggéré d'effectuer la rotation des caractères, nous appellerons « *programme de rotation* » la programme qui permet la programmation de l'heure initiale !

Le but est le suivant : l'entrée \overline{EF}_1 étant activée (mise à 0), la caractéristique correspondant \square_i doit s'incrémenter environ toutes les demi-secondes. Nous utilisons le registre R (B) comme pointeur du caractère \square_i (fig. 33) ; le registre R (6) doit pointer l'opérande correspondant. Le programme de rotation doit commencer par stocker dans l'espace des opérandes, à l'adresse $n' + 2$, l'octet 43 ou 49 selon la valeur du caractère \square_i ; ceci interdit la programmation d'« heures interdites », comme 27 h 52 mn par exemple. Le programme doit ensuite tester l'état des entrées \overline{EF}_1 à \overline{EF}_4 en vue de positionner les pointeurs R (6) et R (B) ; enfin, la rotation du caractère \square_i s'effectue tant que \overline{EF}_1 est activé.

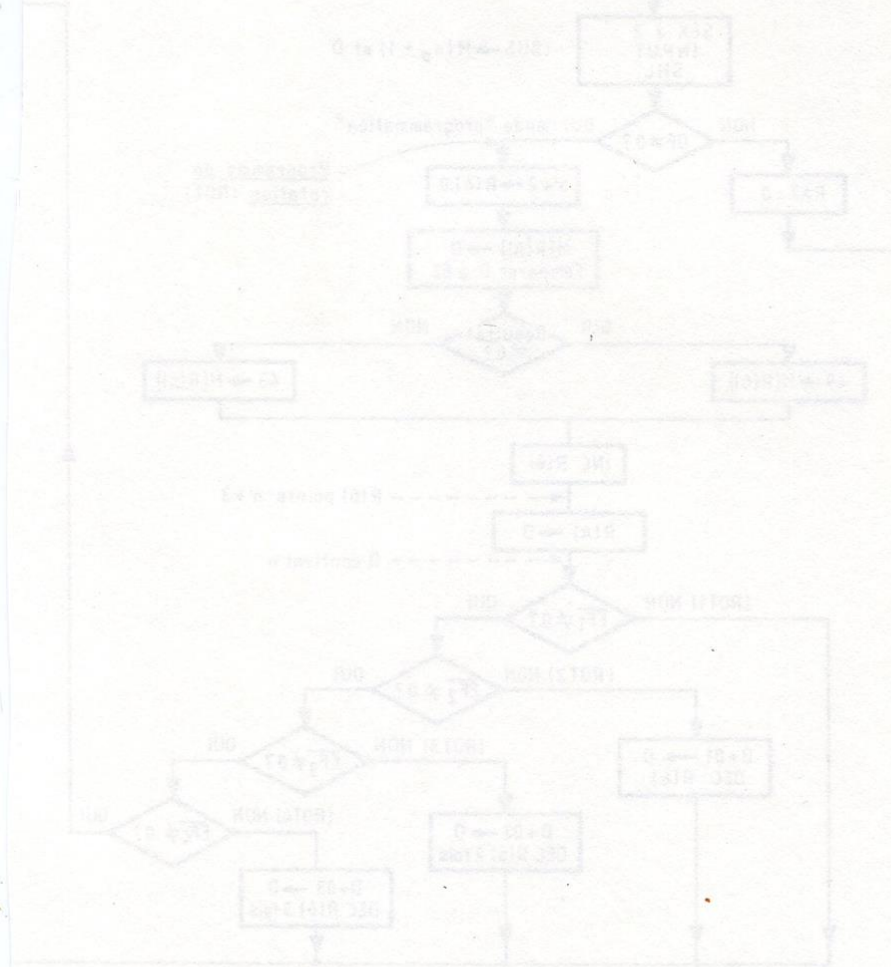
L'organigramme détaillé du programme principal est donné figure 36. Son exécution correcte suppose qu'il est précédé des instructions nécessaires à l'initialisation des compteurs de programme R (1), R (3), R (4) ; à l'initialisation du pointeur de pile R (2) à l'adresse $n_p + 1$; à l'initialisation du pointeur R (A) du caractère \square_1 à l'adresse n ; à l'initialisation du compteur de boucles R (F) par le nombre décimal 60. Pour terminer, il est également au préalable nécessaire de remplir l'espace des opérandes par l'octet 19 à l'adresse n' , l'octet 25 à l'adresse $n' + 1$, l'octet 82 à l'adresse $n' + 3$, ainsi que de stocker dans l'espace d'affichage les octets 80, 40, 20, 10 ; ce dernier point aboutit d'ailleurs à l'affichage de 0000 lors de la mise en route du système.

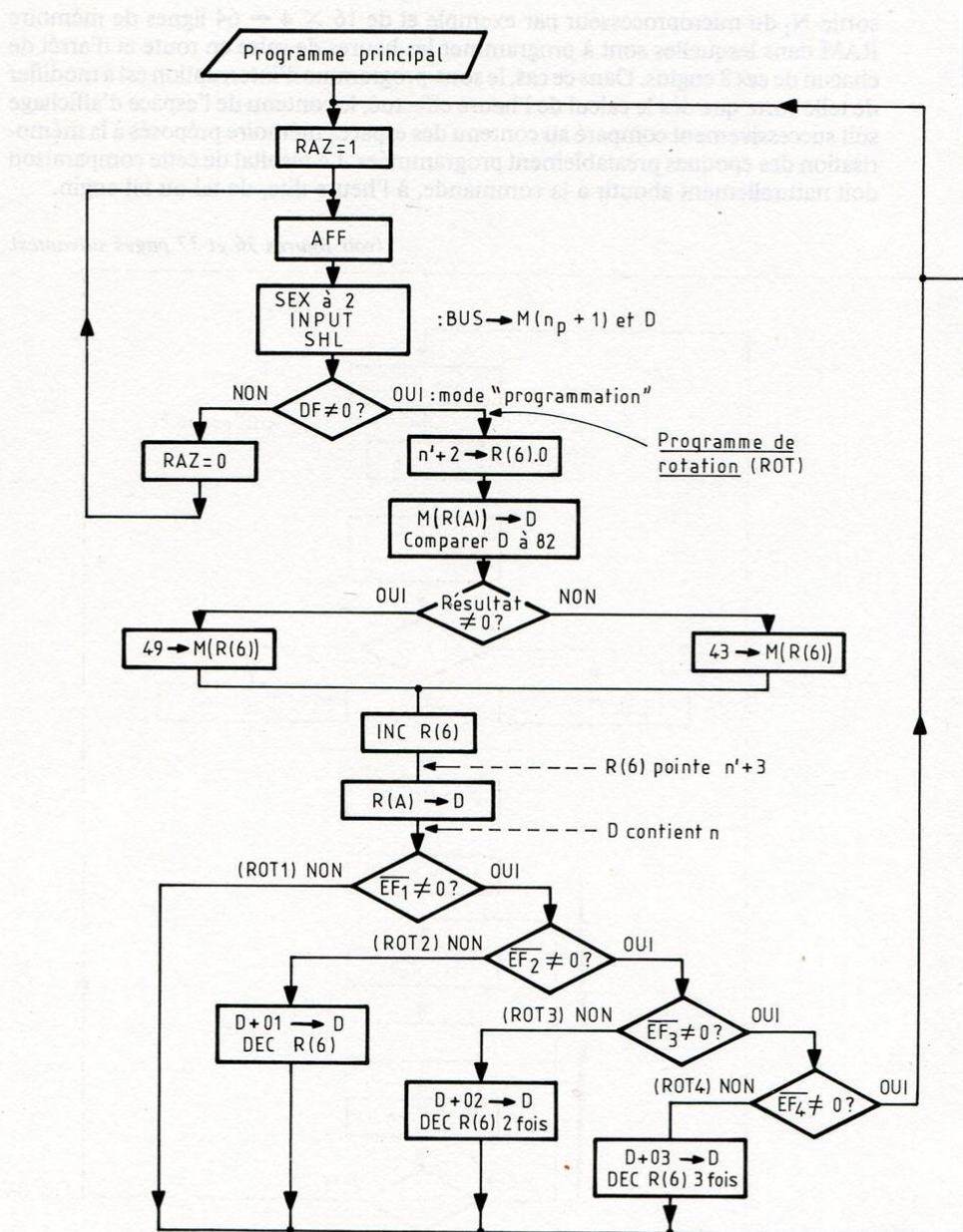
A titre indicatif, nous donnons figure 37 le programme complet de l'horloge. Le fonctionnement est prévu avec un générateur d'interruptions de fréquence 1 Hz.

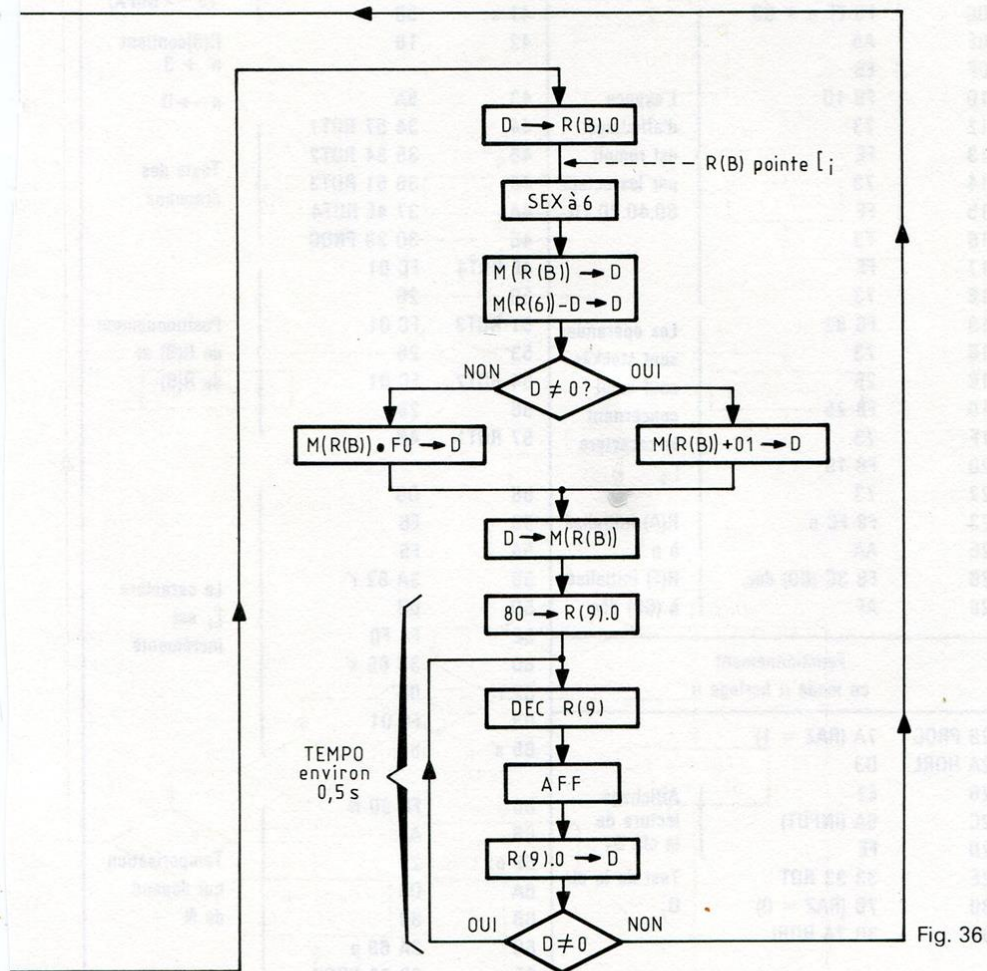
Hormis l'excellent exercice que constitue la mise en œuvre de cette horloge, le lecteur ne verra peut-être pas l'intérêt d'utiliser un microprocesseur pour afficher l'heure exacte. L'intérêt réside dans les extensions que l'on peut facilement donner à notre système. En effet, l'heure qui est affichée est contenue dans les 4 lignes mémoires qui constituent l'espace d'affichage, mais rien n'interdit d'imaginer qu'un certain nombre d'ensembles de 4 lignes mémoire puisse être destiné à mémoriser, dans le mode « programmation », un certain nombre d'époques de déclenchement et d'arrêt d'engins à commander automatiquement. Par exemple, si on veut commander 8 engins, il faut disposer d'un registre de sortie supplémentaire sélectionné par la

sortie N_1 du microprocesseur par exemple et de $16 \times 4 = 64$ lignes de mémoire RAM dans lesquelles sont à programmer les heures de mise en route et d'arrêt de chacun de ces 8 engins. Dans ce cas, le sous-programme d'interruption est à modifier de telle sorte que dès le calcul de l'heure effectué, le contenu de l'espace d'affichage soit successivement comparé au contenu des espaces mémoire préposés à la mémorisation des époques préalablement programmées. Le résultat de cette comparaison doit naturellement aboutir à la commande, à l'heure dite, de tel ou tel engin.

(voir figures 36 et 37 pages suivantes).







Initialisation			Rotation des caractères			
00	F8 71 ENT.INT.	R(1) et R(2) initialisés	33 ROT	F8 FA n' + 2	[₁ est comparé à 2	
02	A1		35	A6		
03	F8 F7 n _p + 1		36	0A		
05	A2	R(3) et R(4) initialisés	37	FD 82	si [₁ = 2, 43 → M(FA) sinon 49 → M(FA)	
06	F8 AF ENT.AFF.		39	3A 3F r		
08	A3		3B	F8 43		
09	F8 BF ENT.TEMP.	L'espace d'affichage est rempli par les octets 80,40,20,10	3D	30 41 s	R(6) contient n' + 3 n → D	
0B	A4		3F r	F8 49		
0C	F8 FF n + 03		41 s	56		
0E	A5	Les opérandes sont stockés sauf celui concernant le caractère [₂	42	16	Tests des drapeaux	
0F	E5		43	8A		
10	F8 10		44	34 57 ROT1		
12	73	R(A) initialisé à n	46	35 54 ROT2	Positionnement de R(B) et de R(6)	
13	FE		48	36 51 ROT3		
14	73		4A	37 4E ROT4		
15	FE	R(F) initialisé à (60) déc.	4C	30 29 PROG	Le caractère [_i est incrémenté	
16	73		4E ROT4	FC 01		
17	FE		50	26		
18	73	Affichage lecture de la clé D ₇ Test de la clé D ₇	51 ROT3	FC 01	Temporisation qui dépend de N	
19	F8 82		53	26		
1B	73		54 ROT2	FC 01		
1C	25	Fonctionnement en mode « horloge »	56	26		
1D	F8 25		57 ROT1	AB		
1F	73		58	0B		
20	F8 19	29 PROG 7A (RAZ = 1)	59	E6		
22	73		5A	F5		
23	F8 FC n		5B	3A 62 r'		
25	AA	2A HORL D3	5D	0B		
26	F8 3C (60) déc.		5E	FA F0		
28	AF		60	30 65 s'		
			62 r'	0B		
			63	FC 01		
			65 s'	5B		
			66	F8 80 N		
			68	A9		
			69 p	29		
			6A	D3		
			6B	89		
			6C	3A 69 p		
			6E	30 29 PROG		

Fig. 37

Sous-programme d'interruption		Sous - programme d'interruption (Suite)	
70 RET.INT.	70	A8 FIN	E2
71 ENT.INT.	22	A9 REST	42
72	78 Sauver T	AA	FE
73	22	AB	42
74	73 Sauver D	AC	30 70 RET
75	76	Restauration du contexte	
76	52 Sauver DF	Sous - programme d'affichage	
77	2F	AE RET.AFF.	D0
78	8F	AF ENT.AFF.	F8 FC n
79	3A A9 REST	B1	A5
7B	F8 3C	B2	F8 04
7D	AF	B4	A7
		B5	E5
7E	F8 FA n' + 2	B6 OUT	61
80	A6	B7	D4
81	0A	B8	27
82	FD 82	B9	87
84	3A 8A r	BA	3A B6 OUT
86	F8 43	BC	30 AE RET.AFF.
88	30 8C s	Sous - programme de temporisation	
8A r	F8 49	BE RET.TEMP	D3
8C s	56	BF ENT.TEMP	F8 20
8D	26	C1	A8
8E	26	C2 DECR.	28
		C3	88
8F	F8 FF (n + 3)	C4	3A C2 DECR
91	A5	C6	30 BE RET.TEMP
92	F8 04		
94	A9		
95	E5		
96 COMP	46		
97	F5		
98	3A A4 INCR.		
9A	F8 F0		
9C	F2		
9D	73		
9E	29		
9F	89		
A0	3A 96 COMP		
A2	30 A8 FIN		
A4 INCR	F8 01		
A6	F4		
A7	55		

Fig. 37

Chapitre 19

LA CONVERSION NUMÉRIQUE \longleftrightarrow ANALOGIQUE

I. La conversion numérique \longrightarrow analogique

I. 1. Principe

Le microprocesseur traite des mots binaires de format 8 bits. Dans grand nombre d'applications il est souhaitable d'être en mesure de transformer un mot binaire en une grandeur électrique analogique (courant ou tension). Cette grandeur électrique est ensuite traduite en une grandeur physique à contrôler, par l'intermédiaire d'un transducteur. Par exemple, on peut demander au microprocesseur de régler la température d'un four destiné à la cuisson du verre ; le transducteur est alors une simple résistance chauffante. La température est contrôlée par le courant circulant dans cette résistance, courant fourni par le microprocesseur par l'intermédiaire d'un convertisseur numérique analogique (CNA). Pour comprendre le principe de fonctionnement d'un CNA, observons la figure 1. Si le potentiel (pris par rapport à la masse) des points D_3, D_2, D_1, D_0 est égal à zéro, il est facile de se rendre compte que :

$$V_3 = -V_R \text{ soit } I_3 = \frac{V_R}{2R} \cdot \frac{1}{2^0}$$

$$V_2 = -\frac{V_R}{2} \text{ soit } I_2 = \frac{V_R}{2R} \cdot \frac{1}{2^1}$$

$$V_1 = -\frac{V_R}{4} \text{ soit } I_1 = \frac{V_R}{2R} \cdot \frac{1}{2^2}$$

$$V_0 = -\frac{V_R}{8} \text{ soit } I_0 = \frac{V_R}{2R} \cdot \frac{1}{2^3}$$

Supposons maintenant (fig. 2), que les points D_3, D_2, D_1, D_0 soient remplacés par des commutateurs commandés par les *signaux logiques* D_3, D_2, D_1, D_0 .

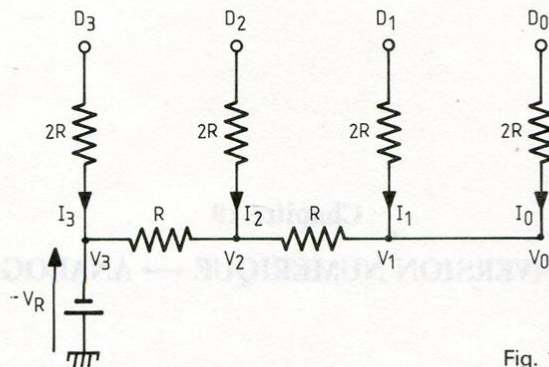


Fig. 1

Ces commutateurs sont en position A si le signal logique D correspondant est au niveau 1, et en position B dans le cas contraire. Les sorties S et \bar{S} (la notation \bar{S} ne veut pas dire ici « complément de S ») sont par ailleurs supposées au potentiel zéro. Les courants I_{OUT} et \bar{I}_{OUT} sont donnés par les expressions suivantes :

$$I_{OUT} = \frac{1}{2^3} \frac{V_R}{2R} [D_3 \cdot 2^3 + D_2 \cdot 2^2 + D_1 \cdot 2^1 + D_0 \cdot 2^0]$$

$$\bar{I}_{OUT} = \frac{1}{2^3} \frac{V_R}{2R} [\bar{D}_3 \cdot 2^3 + \bar{D}_2 \cdot 2^2 + \bar{D}_1 \cdot 2^1 + \bar{D}_0 \cdot 2^0]$$

$$I_{OUT} + \bar{I}_{OUT} = \frac{15}{16} \frac{V_R}{R} = \text{constante}$$

Il faut observer que le mot binaire de 4 bits ($D_3 D_2 D_1 D_0$) est transformé en la grandeur analogique I_{OUT} , image de ce mot. La notation \bar{I}_{OUT} veut dire que \bar{I}_{OUT} est l'image du mot ($\bar{D}_3 \bar{D}_2 \bar{D}_1 \bar{D}_0$).

L'ensemble de la figure 2 constitue un CNA dont la grandeur de sortie I_{OUT} peut aisément être transformée en tension U_{OUT} grâce à l'adjonction d'un amplificateur opérationnel comme l'indique la figure 3. On vérifie aisément que les points S et \bar{S} sont au potentiel zéro, et que $U_{OUT} = R \cdot I_{OUT}$. Lorsque ($D_3 D_2 D_1 D_0$) = (0000), $U_{OUT} = 0$ et lorsque ($D_3 D_2 D_1 D_0$) = (1111), $U_{OUT} = \frac{15}{16} V_{REF}$.

Il faut noter que si la sortie \bar{I}_{OUT} du CNA est reliée à l'entrée inverseuse de l'amplificateur opérationnel, et si I_{OUT} est relié à la masse, nous avons $U_{OUT} = R \cdot \bar{I}_{OUT}$. Dans ce cas, $U_{OUT} = 0$ lorsque ($\bar{D}_3 \bar{D}_2 \bar{D}_1 \bar{D}_0$) = (0000) et $U_{OUT} = \frac{15}{16} V_{REF}$ lorsque ($\bar{D}_3 \bar{D}_2 \bar{D}_1 \bar{D}_0$) = (1111).

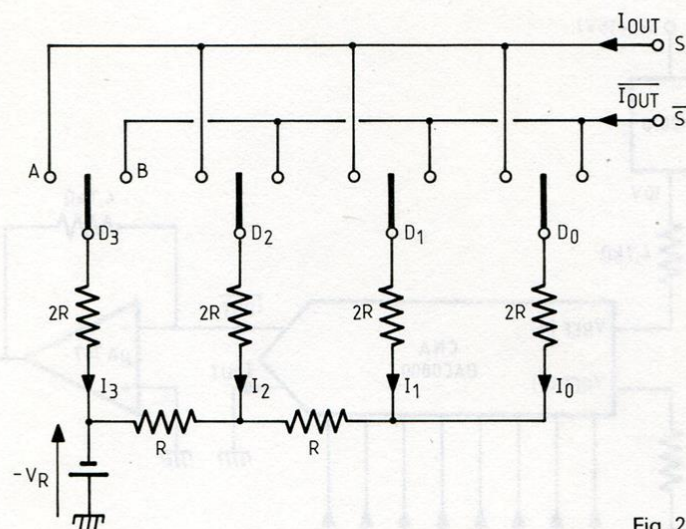


Fig. 2

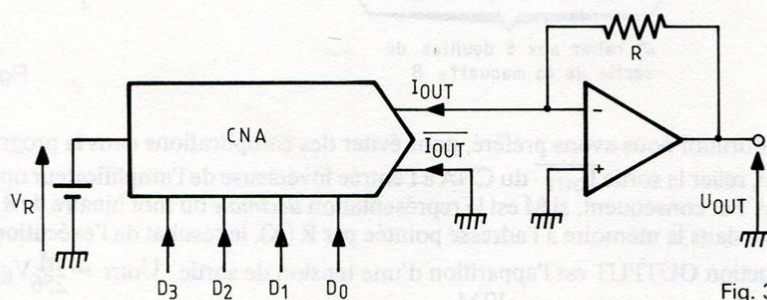


Fig. 3

Le lecteur possesseur de l'ensemble maquette A – maquette B, désirant disposer d'une grandeur analogique de sortie, pourra réaliser le circuit d'interface de la figure 4. Ce circuit nécessite une alimentation double ($\pm 15V$) et permet de disposer d'une tension de sortie U_{OUT} variant entre 0 et $\frac{255}{256} V_{REF}$, c'est-à-dire pratiquement de 0 à 10 V si la tension de référence est choisie égale à 10V. Les 8 entrées logiques du CNA sont à relier aux 8 douilles de la maquette B correspondant aux sorties DO7 à DO0 du registre CDP 1852. Notez que ces douilles supportent en réalité les grandeurs complémentées \overline{DO}_7 à \overline{DO}_0 à cause des transistors de commande des LED ;

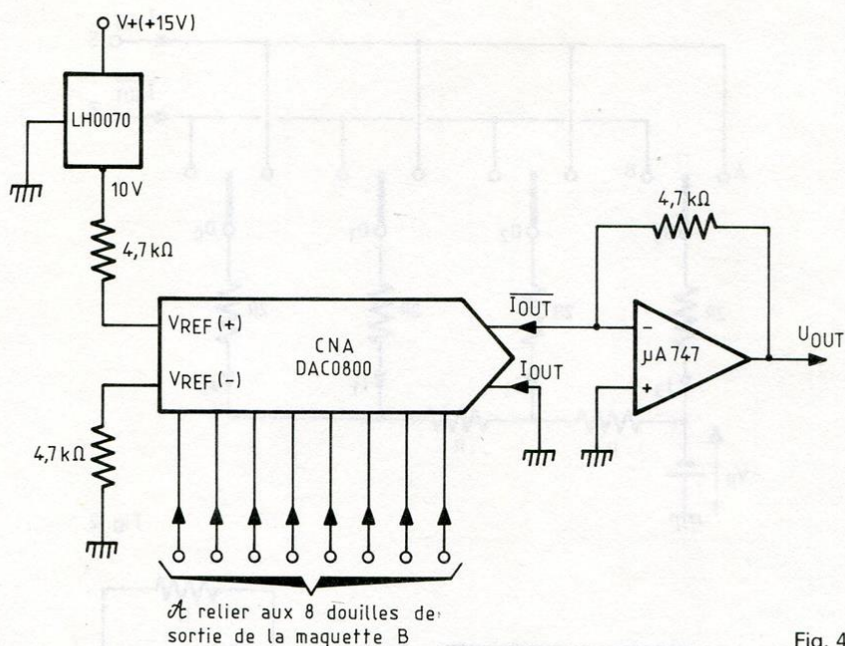


Fig. 4

c'est pourquoi nous avons préféré, pour éviter des complications dans la programmation, relier la sortie $\overline{I_{OUT}}$ du CNA à l'entrée inverseuse de l'amplificateur opérationnel. Par conséquent, si M est la représentation *décimale* du mot binaire de 8 bits contenu dans la mémoire à l'adresse pointée par $R(X)$, le résultat de l'exécution de l'instruction OUTPUT est l'apparition d'une tension de sortie $U_{OUT} = \frac{M}{256} V_{REF}$, c'est-à-dire, en volts, $U_{OUT} = \frac{10M}{256}$

I. 2. Exemple d'application

Le circuit d'interface que nous venons de décrire peut être destiné à contrôler la déviation du faisceau d'électrons émis par le canon d'un oscilloscope. Pour l'application qui nous intéresse, la sortie U_{OUT} est reliée à l'une des voies d'entrée de l'oscilloscope dont nous utilisons la base de temps interne. Cependant, nous demanderons à la sortie Q du microprocesseur de se charger de la synchronisation, c'est-à-dire du déclenchement de la base de temps.

Supposez qu'une zone mémoire, que nous appellerons « espace d'affichage », contienne la suite des mots binaires M_1, M_2, \dots, M_k (fig. 5). L'adresse du haut de l'es-

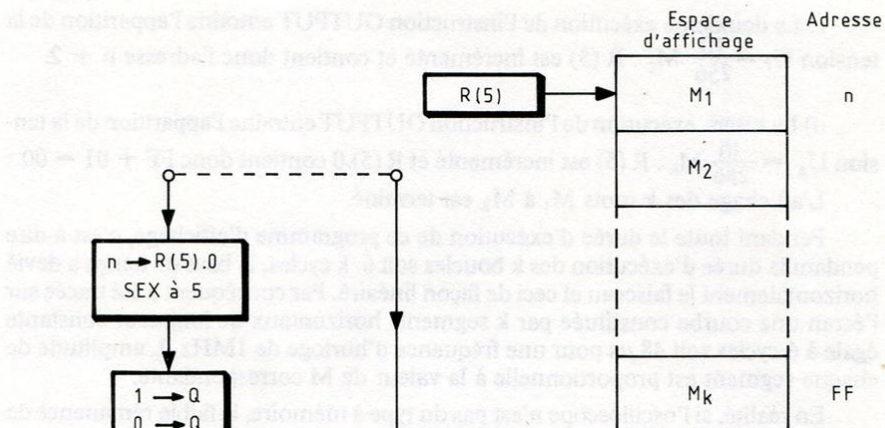


Fig. 5

Fig. 6

pace d'affichage est n , l'adresse du bas est FF ; par ailleurs, cet espace est pointé par le registre $R(5)$ par exemple. Quel est le résultat de l'exécution du programme dont l'organigramme est donné figure 6 ?

a) Le pointeur $R(5)$ étant initialisé à n , l'impulsion sur la sortie Q reliée à l'entrée « Synchro Extérieure » de l'oscilloscope, a pour effet de déclencher la base de temps.

b) La première exécution de l'instruction **OUTPUT** entraîne l'apparition, sur la voie d'entrée de l'oscilloscope, de la tension $U_1 = \frac{10}{256} M_1$. $R(5)$ est incrémenté et contient donc l'adresse $n + 1$.

c) La deuxième exécution de l'instruction OUTPUT entraîne l'apparition de la tension $U_2 = \frac{10}{256} M_2 \cdot R(5)$ est incrémenté et contient donc l'adresse $n + 2$.

d) La $k^{\text{ième}}$ exécution de l'instruction OUTPUT entraîne l'apparition de la tension $U_k = \frac{10}{256} M_k \cdot R(5)$ est incrémenté et $R(5).0$ contient donc $FF + 01 = 00$:

L'affichage des k mots M_1 à M_k est terminé.

Pendant toute la durée d'exécution de ce programme d'affichage, c'est-à-dire pendant la durée d'exécution des k boucles soit 6. k cycles, la base de temps a dévié horizontalement le faisceau et ceci de façon linéaire. Par conséquent a été tracée sur l'écran une courbe constituée par k segments horizontaux de longueur constante égale à 6 cycles soit $48 \mu s$ pour une fréquence d'horloge de 1MHz. L'amplitude de chaque segment est proportionnelle à la valeur de M correspondante.

En réalité, si l'oscilloscope n'est pas du type à mémoire, la faible rémanence de l'écran nécessite le rafraîchissement de l'affichage. (fig. 7).

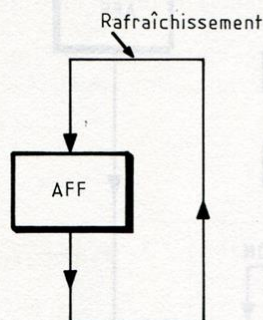


Fig. 7

Proposons-nous le but suivant : une population est constituée par un certain nombre d'individus dont les âges sont compris entre 1 an et 20 ans. Nous désirons que chaque individu se présente devant le système à microprocesseur, affiche son âge par l'intermédiaire des clés de la maquette A, puis l'introduise en enfonçant une touche que nous appellerons touche ENT. Le microprocesseur doit à tout instant afficher sur l'écran de l'oscilloscope la *courbe de distribution des âges*, c'est-à-dire la courbe telle que les coordonnées du point figuratif sont :

- pour l'abscisse, l'âge
- pour l'ordonnée, le nombre d'individus ayant cet âge.

Lorsqu'aucun individu ne s'est encore présenté, la courbe de distribution doit être réduite à une droite horizontale d'ordonnée zéro. Par conséquent, les premières instructions du programme doivent être destinées à remplir tout l'espace d'affichage

de zéros. L'effet de l'enfoncement de la touche ENT doit être le stockage, dans l'espace d'affichage, de l'information correspondant à la donnée présentée sur les clés de la maquette A. Dès que ce stockage est opéré, le microprocesseur doit afficher la nouvelle courbe de distribution. Comme l'affichage doit être constamment rafraîchi, nous proposons que l'enfoncement de la touche ENT ait pour résultat d'appeler un sous-programme d'interruption destiné à exécuter toutes les instructions utiles à l'introduction dans le système, de l'âge de l'individu concerné. La touche ENT est donc le bouton poussoir de la maquette A si la sortie \overline{SR} de la bascule JK est reliée à l'entrée $\overline{INTERRUPT}$ du microprocesseur.

L'espace mémoire RAM nécessaire est organisé comme suit : (fig. 8)

- la pile sauvegarde les contenus de D, X et P lors de l'exécution du sous-programme d'interruption ; par ailleurs, la ligne d'adresse $n_p - 2$ contient le mot présent sur le bus des données lors de l'instruction INPUT.

- l'espace d'affichage est constitué des 20 lignes d'adresses EC à FF. La ligne d'adresse EC doit contenir le nombre d'individus ayant 1 an, la ligne d'adresse ED le nombre d'individus ayant 2 ans, ..., la ligne d'adresse FF le nombre d'individus ayant 20 ans. L'organigramme du programme principal est donné figure 9, et celui du sous-programme d'interruption figure 10. Celui-ci appelle les commentaires suivants.

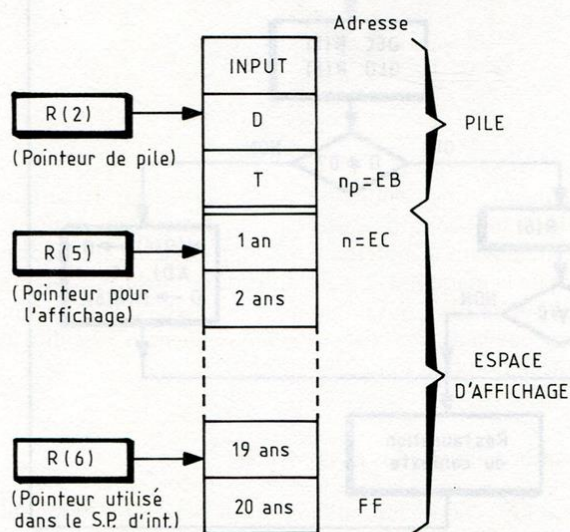


Fig. 8

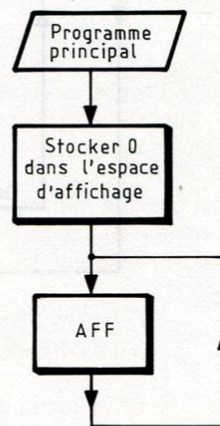


Fig. 9

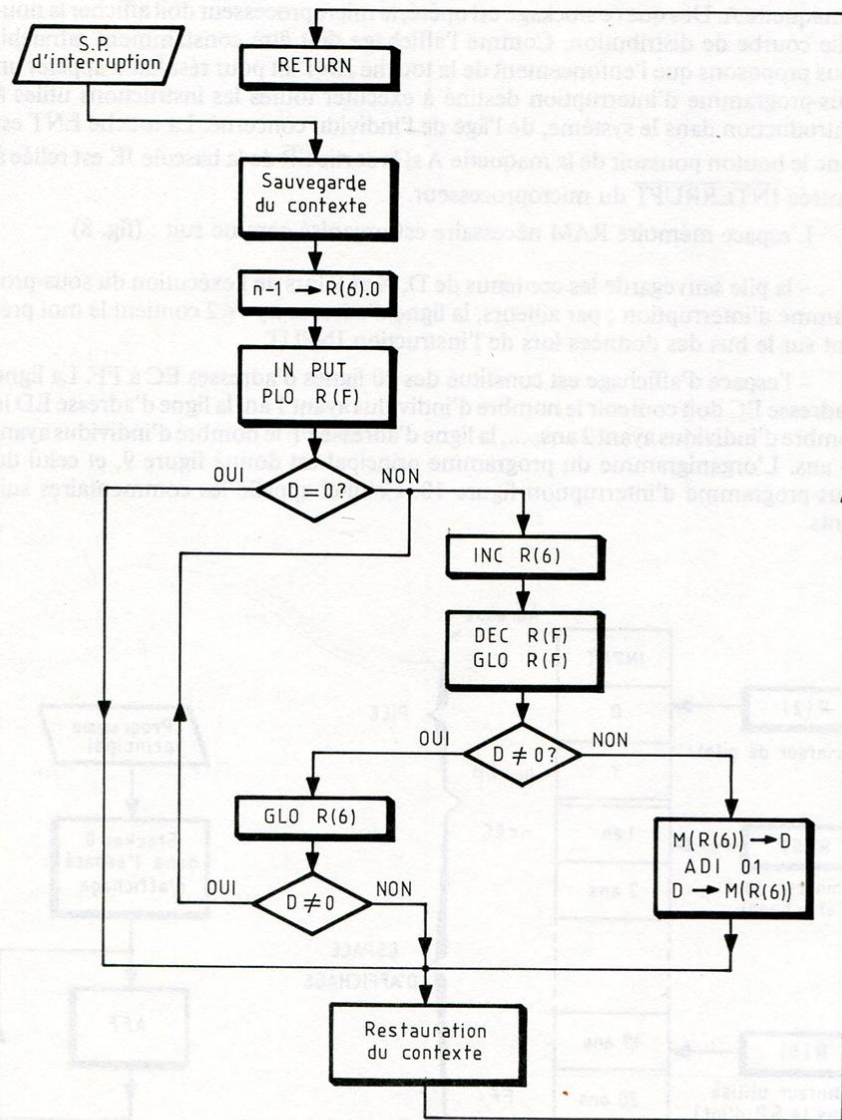


Fig. 10

Initialisation			S.P. d'interruption		
00	F8 21 ENT.INT.	LDI 21	20 RET.INT. 70		RET
02	A1	PLO R(1)	21 ENT.INT. 22		DEC R(2)
03	F8 EC $n_p + 1$	LDI $n_p + 1$	22	78	SAV T
05	A2	PLO R(2)	23	22	DEC R(2)
			24	73	STXD (SAV D)
	<u>Stocker 00 dans l'espace d'affichage</u>		25	F8 EB n - 1	LDI n - 1
06	F8 FF	LDI FF	27	A6	PLO R(6)
08	A5	PLO R(5)	28	6A	INP
09	E5	SEX	29	AF	PLO R(F)
0A	F8 14 (20 déc.)	LDI (20) déc.	2A	32 3A REST.	BZ REST
0C	AA	PLO R(A)	2C INCR.	16	INC R(6)
0D CLEAR	FA 00	ANI 00	2D	2F	DEC R(F)
0F	73	STXD	2E	8F	GLO R(F)
10	2A	DEC R(A)	2F	3A 37 RECH.	BNZ RECH.
11	8A	GLO R(A)	31	06	LD R(6)
12	3A 0D CLEAR	BNZ CLEAR	32	FC 01	ADI 01
			34	56	STR M(R(6))
	<u>Affichage</u>		35	30 3A REST.	BR REST.
14 AFF	F8 EC n	LDI n	37 RECH.	86	GLO R(6)
16	A5	PLO R(5)	38	3A 2C INCR	BNZ INCR
17	E5	SEX	3A REST.	12	INC R(2)
18	7B	SEQ	3B	42	LDA M (R(2))
19	7A	REQ	3C	30 20 RET.INT.	BR RET. INT.
1A OUT	61	OUT			
1B	85	GLO R(5)			
1C	3A 1A OUT	BNZ OUT			
1E	30 14 AFF	BR AFF			

Fig. 11

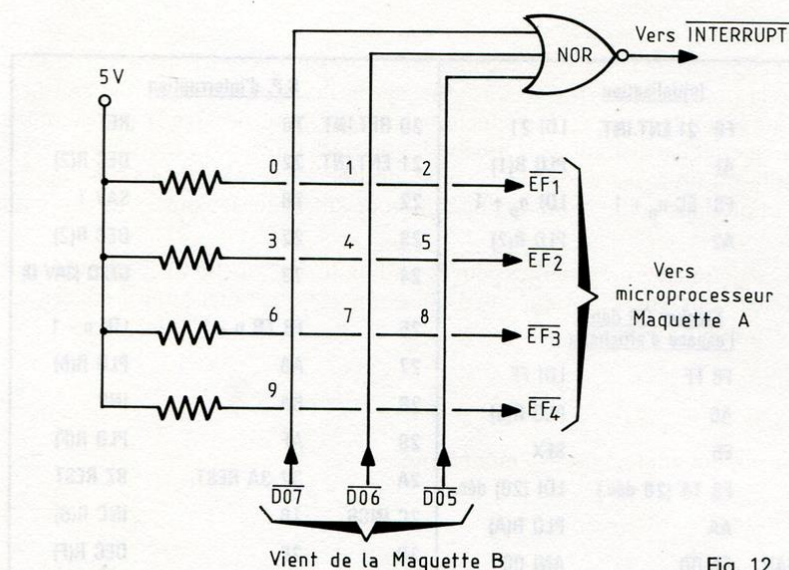


Fig. 12

– Après la procédure de sauvegarde du contexte, le registre R(2) pointe la ligne d'adresse $n_p - 2$. Lors de l'exécution de l'instruction INPUT, le mot présent sur les clés est donc mémorisé à l'adresse $n_p - 2$, ce dont nous ne nous servons point. Il est également transféré dans R(F).0. Si ce mot est nul, il ne se passe rien, c'est-à-dire que l'âge « 0 an » n'est pas pris en compte.

– R(6).0 étant initialisé à $n - 1$, la boucle dans laquelle ce registre est incrémenté et R(F) décrémenté est une procédure de recherche de la ligne mémoire de l'espace d'affichage à modifier. Si la représentation décimale du mot binaire de 8 bits présenté sur les clés est supérieure à 20, il ne se passe rien, c'est-à-dire que les âges supérieurs à 20 ne sont pas non plus pris en compte

– R(6) pointant la ligne correcte, M(R(6)) est incrémenté.

Le programme complet de ce « traceur de courbe de distribution » est donné figure 11. Il peut naturellement être amélioré et étendu à d'autres applications du même genre.

Par exemple, suggérons les idées suivantes :

– Réfléchir à un programme permettant d'entrer les données (les âges) par l'intermédiaire d'un clavier. Si ce clavier est constitué par 10 touches numérotées de 0 à 9, il peut présenter la structure matricielle de la figure 12. ($\overline{D07}, \overline{D06}, \overline{D05}$) étant égal à (0, 0, 0), c'est l'enfoncement d'une touche qui provoque l'interruption de program-

me par l'intermédiaire de la porte NOR à 3 entrées. Le sous-programme d'interruption a donc également la tâche de décoder le clavier. Par ailleurs, l'opération de stockage dans la zone mémoire réservée à l'affichage ne doit s'opérer qu'après l'enfoncement d'une seconde touche (l'introduction dans le système de l'âge 18 par exemple, nécessite en effet l'activation de la touche 1, puis de la touche 8) ; ceci peut être réalisé en utilisant un registre R initialisé à 02, destiné à orienter le sous-programme d'interruption dans la direction de l'opération de stockage, seulement une fois sur deux.

– Aux 10 touches précédentes peut être adjointe une touche CLEAR utile dans le cas d'une erreur dans l'introduction d'une donnée. Cette touche doit « effacer » la donnée précédemment introduite.

II. La conversion analogique \rightarrow numérique

II. 1. Principe

Dans grand nombre d'applications, il est souhaitable d'être en mesure de transformer la grandeur électrique analogique (courant ou tension) fournie par un transducteur, en une grandeur numérique (un mot binaire de 8 bits). En effet, cette grandeur numérique, image de la grandeur analogique, peut alors aisément être traitée par le microprocesseur. Par exemple, si l'on désire réguler la température d'un four, il est au préalable nécessaire de mesurer ses variations éventuelles. Dans ce cas, le transducteur peut être une diode au silicium polarisée en direct. La température peut être introduite dans le microprocesseur par l'intermédiaire d'un convertisseur analogique numérique (CAN) destiné à transformer la différence de potentiel aux bornes de la diode en son image numérique. Pour comprendre le principe de fonctionnement d'un CAN, examinons la figure 13.

La tension U à mesurer, supposée comprise entre 0 et V_{REF} , est reliée à l'entrée inverseuse d'un comparateur susceptible de fournir un signal logique F.C. (Fin de Conversion) égal à 0 si la tension de sortie U_{OUT} du CNA est inférieure à U , et égal à 1 dans le cas contraire. Le mot binaire de 8 bits, placé à l'entrée du CNA, est fourni par un compteur déclenché par l'horloge si la sortie \overline{Q} de la bascule est au niveau 0 ; si la sortie \overline{Q} de la bascule est au niveau 1, le compteur est bloqué. Lorsque le compteur est bloqué, le mot binaire présent en sa sortie peut être mémorisé dans le coupleur (registre CDF 1852 par exemple), puis transféré sur le bus des données du microprocesseur. Ces deux opérations sont effectuées par l'intermédiaire respectivement des entrées CL (mémorisation) et CS (transfert) du coupleur.

Explicitons le fonctionnement d'un tel système en supposant qu'au début de notre étude, la sortie \overline{Q} de la bascule est au niveau 1 (le compteur est bloqué), la ten-

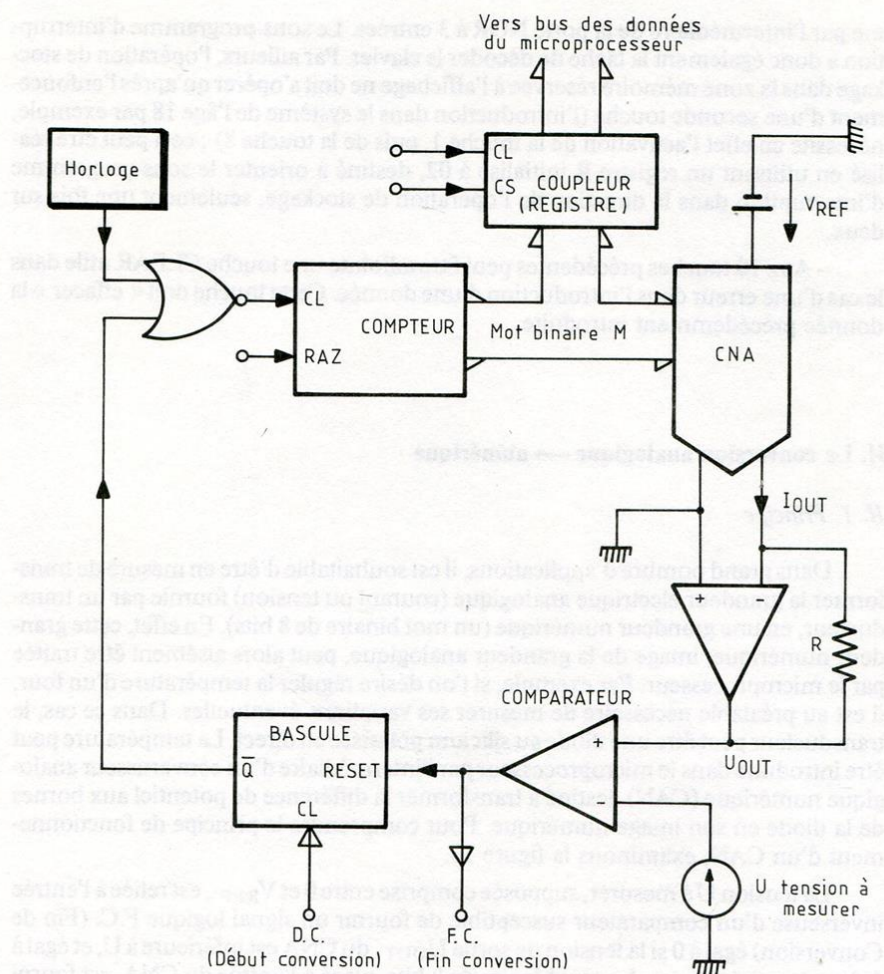


Fig. 13

sion de sortie U_{OUT} du CNA est égale à 0 (le mot binaire M est égal à zéro), l'entrée RAZ du compteur n'est pas activée :

a) La sortie F.C. du comparateur étant au niveau 0, une impulsion sur l'entrée CL de la bascule a pour effet de déclencher le compteur : La conversion commence.

b) Pendant l'incrémentation du mot M , U_{OUT} croît linéairement à partir de 0 et constitue une rampe de tension. Si F est la fréquence de l'horloge, l'équation de cette rampe est donnée par l'expression $U_{OUT} = \frac{V_{REF}}{256} \cdot F \cdot t$.

c) Lorsque U_{OUT} atteint la valeur U , c'est-à-dire au bout d'un temps t_c (temps de conversion) égal à $t_c = \frac{U \cdot 256}{V_{REF} \cdot F}$, la sortie F.C. du comparateur passe au niveau 1,

indiquant par ce fait que la conversion est terminée. Ceci a pour effet de remettre à zéro la bascule, autrement dit de remettre sa sortie \bar{Q} au niveau 1, bloquant ainsi le compteur.

d) A ce moment, le mot binaire M est l'image numérique de la grandeur analogique U . Si M est la représentation décimale de ce mot, nous avons $M = 256 \frac{U}{V_{REF}}$. Il ne reste plus qu'à le transférer vers le microprocesseur. Notez qu'une nouvelle conversion nécessite l'activation du signal RAZ du compteur.

Remarques :

- Contrairement à la conversion numérique analogique, la conversion analogique numérique prend du temps. Par exemple, si $F = 1\text{MHz}$, la conversion demande $256\mu\text{s}$ si U a la valeur maximale V_{REF} . Pendant toute cette durée, il est nécessaire que U ne varie pas. Si c'est cependant le cas, il y a lieu d'échantillonner la tension U variable de telle sorte à la transformer en une succession de paliers de largeur au moins égale au temps de conversion (fig. 14). Nous ne développerons pas ces considérations.

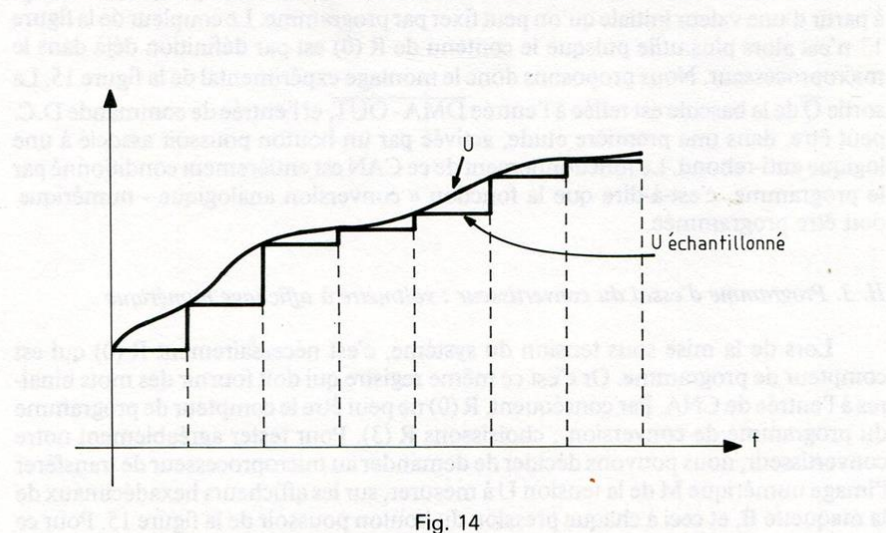


Fig. 14

- On peut notablement gagner du temps en remplaçant le compteur par un registre fournissant des mots binaires M selon la séquence suivante; pour clarifier le langage, nous noterons M_0 le mot image de la tension U à mesurer :

- a) $M = 80$ (Hexa) est placé en entrée du CNA.
Si $U_{OUT} > U$ (indiqué par le comparateur), $M > M_0$ et si $U_{OUT} < U$, $M < M_0$.
- b) Selon les deux cas précédents, respectivement $M = 40$ ou $M = C0$ est placé en entrée du CNA, d'où il résulte de nouveau deux cas pour chacune de ces valeurs.
- c) Le processus est réitéré jusqu'à temps que, par approximations successives, M_0 soit trouvé.

On remarque ainsi que ce procédé conduit à l'encadrement de M_0 par des « fenêtres » toujours deux fois plus étroites que les précédentes. Si par exemple $M = F2$, ce type de registre fournit en entrée du CNA la suite des mots 80, C0, EO, FO, F8, F4, F2, ce qui a nécessité seulement 7 impulsions d'horloge. Observez qu'un compteur fournit le mot F2 au bout de 242 impulsions d'horloge !

II. 2. Interfaçage avec le microprocesseur CDP 1802

Revenons à la figure 13. L'amateur désireux de réaliser un CAN peut a priori être rebuté par la complexité du montage. Cependant, la structure matérielle peut être considérablement simplifiée si on se rend compte que le microprocesseur contient un compteur tout à fait adapté à la fonction qui nous intéresse : il s'agit du registre $R(0)$ qui a la propriété de s'incrémenter automatiquement dans le mode DMA. Si par exemple l'entrée DMA-OUT est imposée au niveau 0, le contenu de $R(0)$ apparaissant sur le bus des adresses est incrémenté toutes les 8 périodes d'horloge à partir d'une valeur initiale qu'on peut fixer par programme. Le coupleur de la figure 13 n'est alors plus utile puisque le contenu de $R(0)$ est par définition déjà dans le microprocesseur. Nous proposons donc le montage expérimental de la figure 15. La sortie \overline{Q} de la bascule est reliée à l'entrée $\overline{DMA-OUT}$, et l'entrée de commande D.C. peut être, dans une première étude, activée par un bouton poussoir associé à une logique anti-rebond. Le fonctionnement de ce CAN est entièrement conditionné par le programme, c'est-à-dire que la fonction « conversion analogique - numérique doit être programmée.

II. 3. Programme d'essai du convertisseur : voltmètre à affichage numérique

Lors de la mise sous tension du système, c'est nécessairement $R(0)$ qui est compteur de programme. Or c'est ce même registre qui doit fournir des mots binaires à l'entrée de CNA. Par conséquent, $R(0)$ ne peut être le compteur de programme du programme de conversion ; choisissons $R(3)$. Pour tester agréablement notre convertisseur, nous pouvons décider de demander au microprocesseur de transférer l'image numérique M de la tension U à mesurer, sur les afficheurs hexadécimaux de la maquette B, et ceci à chaque pression du bouton poussoir de la figure 15. Pour ce

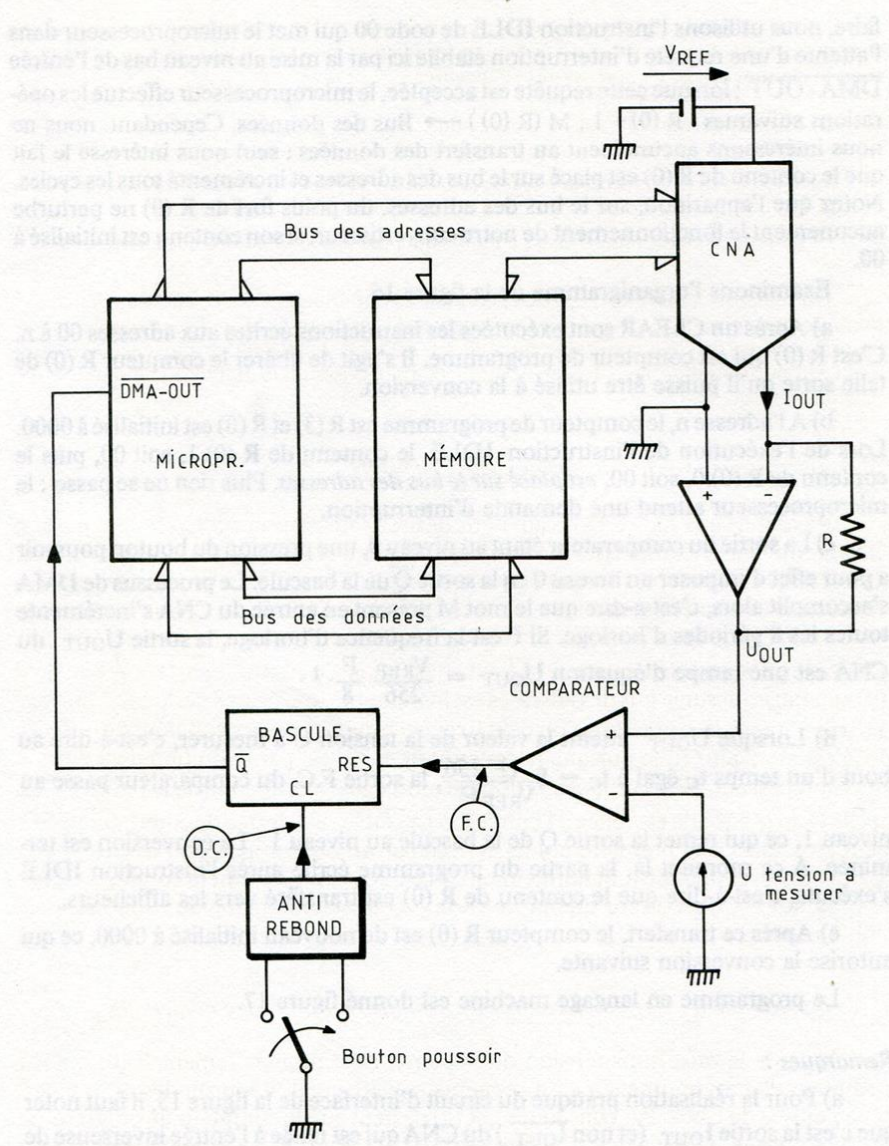


Fig: 15

faire, nous utilisons l'instruction IDLE de code 00 qui met le microprocesseur dans l'attente d'une requête d'interruption établie ici par la mise au niveau bas de l'entrée $\overline{\text{DMA-OUT}}$; lorsque cette requête est acceptée, le microprocesseur effectue les opérations suivantes : $R(0) + 1 : M(R(0)) \rightarrow \text{Bus des données}$. Cependant, nous ne nous intéressons aucunement au transfert des données ; seul nous intéresse le fait que le contenu de $R(0)$ est placé sur le bus des adresses et incrémenté tous les cycles. Notez que l'apparition, sur le bus des adresses, du poids fort de $R(0)$ ne perturbe aucunement le fonctionnement de notre convertisseur, si son contenu est initialisé à 00.

Examinons l'organigramme de la figure 16 :

a) Après un CLEAR sont exécutées les instructions écrites aux adresses 00 à n. C'est $R(0)$ qui est compteur de programme. Il s'agit de libérer le compteur $R(0)$ de telle sorte qu'il puisse être utilisé à la conversion.

b) A l'adresse n, le compteur de programme est $R(3)$ et $R(0)$ est initialisé à 0000. Lors de l'exécution de l'instruction IDLE, le contenu de $R(0).1$, soit 00, puis le contenu de $R(0).0$, soit 00, est placé sur le bus des adresses. Plus rien ne se passe : le microprocesseur attend une demande d'interruption.

c) La sortie du comparateur étant au niveau 0, une pression du bouton poussoir a pour effet d'imposer un niveau 0 en la sortie \overline{Q} de la bascule. Le processus de DMA s'accomplit alors, c'est-à-dire que le mot M présent en entrée du CNA s'incrémente toutes les 8 périodes d'horloge. Si F est la fréquence d'horloge, la sortie U_{OUT} du CNA est une rampe d'équation $U_{\text{OUT}} = \frac{V_{\text{REF}}}{256} \frac{F}{8} \cdot t$.

d) Lorsque U_{OUT} atteint la valeur de la tension U à mesurer, c'est-à-dire au bout d'un temps t_c égal à $t_c = 8 \frac{U}{V_{\text{REF}}} \frac{256}{F}$, la sortie F.C. du comparateur passe au

niveau 1, ce qui remet la sortie Q de la bascule au niveau 1 : La conversion est terminée. A ce moment là, la partie du programme écrite après l'instruction IDLE s'exécute, c'est-à-dire que le contenu de $R(0)$ est transféré vers les afficheurs.

e) Après ce transfert, le compteur $R(0)$ est de nouveau initialisé à 0000, ce qui autorise la conversion suivante.

Le programme en langage machine est donné figure 17.

Remarques :

a) Pour la réalisation pratique du circuit d'interface de la figure 15, il faut noter que c'est la sortie I_{OUT} (et non $\overline{I_{\text{OUT}}}$) du CNA qui est reliée à l'entrée inverseuse de l'amplificateur opérationnel. Par ailleurs, l'entrée $\overline{\text{DMA-OUT}}$ du microprocesseur est utilisée ; or elle n'est pas disponible sur la maquette A. On peut cependant réaliser le convertisseur en reliant la sortie \overline{Q} de la bascule à l'entrée $\overline{\text{DMA-IN}}$; dans ce cas il

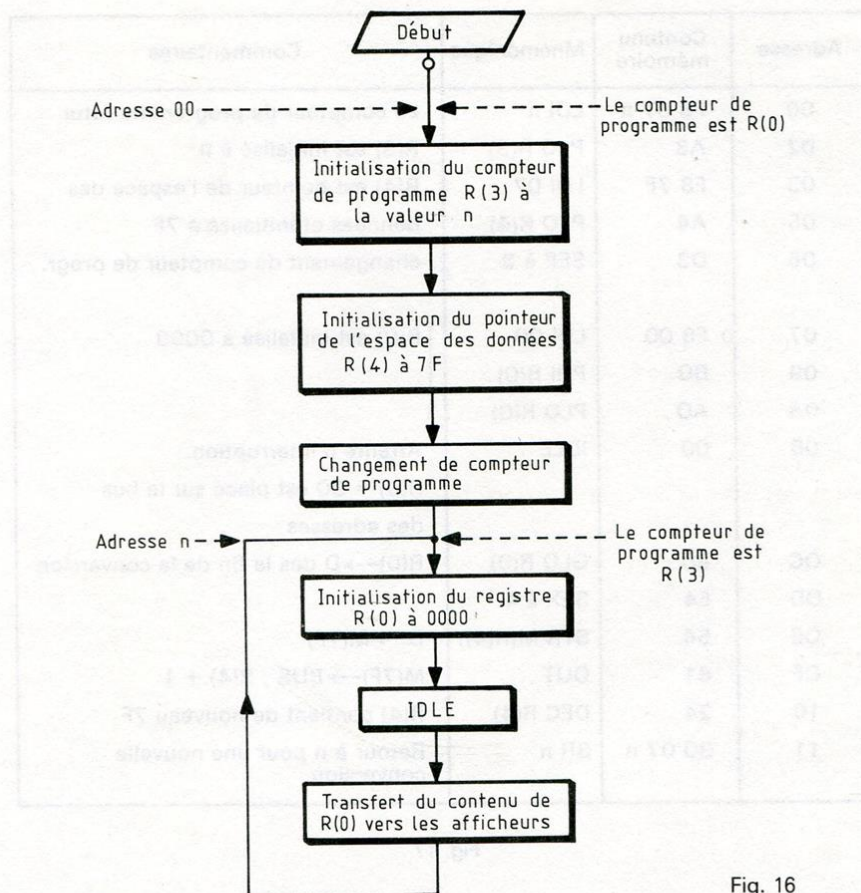


Fig. 16

faut interdire l'écriture de la mémoire pendant l'exécution du processus DMA. Ceci ne peut être réalisé manuellement en plaçant l'inverseur de la maquette B en position « ROM », puisqu'alors il ne serait pas possible de stocker le contenu de R(0) dans la mémoire en vue de son affichage, à moins que l'horloge ne soit réglée à une très basse fréquence (ou en pas-à-pas). A fréquence d'horloge élevée, l'écriture dans la mémoire lors du processus DMA peut être inhibée en commandant $\overline{\text{MWR}}$ (disponible sur le bornier) par le complément du signal DMA - IN.

b) Pour le programme donné figure 17, c'est une pression du bouton poussoir de la figure 15 qui commande l'exécution de la mesure de la tension U. Cette mesure

Adresse	Contenu mémoire	Mnémonique	Commentaires
00	F8 07 n	LDI n	Le compteur de programme futur R(3) est initialisé à n. R(4) est pointeur de l'espace des données et initialisé à 7F changement de compteur de progr.
02	A3	PLO R(3)	
03	F8 7F	LDI 07	
05	A4	PLO R(4)	
06	D3	SEP à 3	
07	n F8 00	LDI 00	R(0) est initialisé à 0000 Attente d'interruption : R(0) = 00 est placé sur le bus des adresses
09	B0	PHI R(0)	
0A	A0	PLO R(0)	
0B	00	IDLE	
0C	80	GLO R(0)	R(0) → D dès la fin de la conversion
0D	E4	SEX à 4	
0E	54	STR M(R(4))	D → M(7F)
0F	61	OUT	M(7F) → BUS ; R(4) + 1
10	24	DEC R(4)	R(4) contient de nouveau 7F
11	30 07 n	BR n	Retour à n pour une nouvelle conversion

Fig. 17

est automatiquement effectuée si on remplace l'instruction IDLE par l'instruction INPUT de code 6 A qui active la sortie N_1 du microprocesseur. Cette sortie doit alors être reliée à l'entrée CL de la bascule.

c) L'avantage essentiel de notre convertisseur est la simplicité de sa structure matérielle. Cet avantage est contrebalancé par un défaut gênant dans le cas où la tension U est rapidement variable. En effet, le temps de conversion est 8 fois plus important que celui du convertisseur réalisé selon le principe donné figure 13. Pour une fréquence F d'horloge de 1MHz par exemple, le temps de conversion est égal à 2 ms lorsque U a sa valeur maximale V_{REF} (au lieu de 250 μ s).

Chapitre 20

LES MATÉRIELS UTILISÉS

En dehors des mémoires dont une étude particulière a été entreprise dès le début de cet ouvrage, automates et microprocesseurs utilisent des éléments classiques de la logique combinatoire (multiplexeurs, décodeurs, ALU) et de la logique séquentielle (bascules, registres, compteurs). Afin d'aider éventuellement la lecture de ce livre, nous rappelons ici succinctement *la fonction* de ces éléments, sans nous préoccuper (sauf exception) de leur structure interne. Pour plus de précision, il est conseillé de se reporter à un ouvrage de base de logique.

I. Eléments de la logique combinatoire

I. 1. Multiplexeurs

Dans l'exemple de la figure 1 (multiplexeur 2 vers 1) :

$S = E_0$ si et seulement si $A_0 = 0$

$S = E_1$ si et seulement si $A_0 = 1$

Ce multiplexeur est donc un système permettant d'aiguiller vers la sortie S l'entrée E_0 ou l'entrée E_1 selon l'état de la commande A_0 (adresse).

L'équation logique de ce circuit est : $S = E_0 \cdot \overline{A_0} \vee E_1 \cdot A_0$.

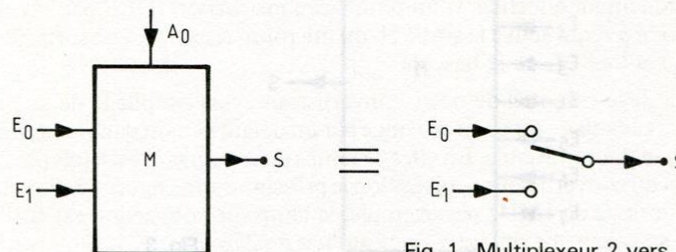


Fig. 1. Multiplexeur 2 vers 1

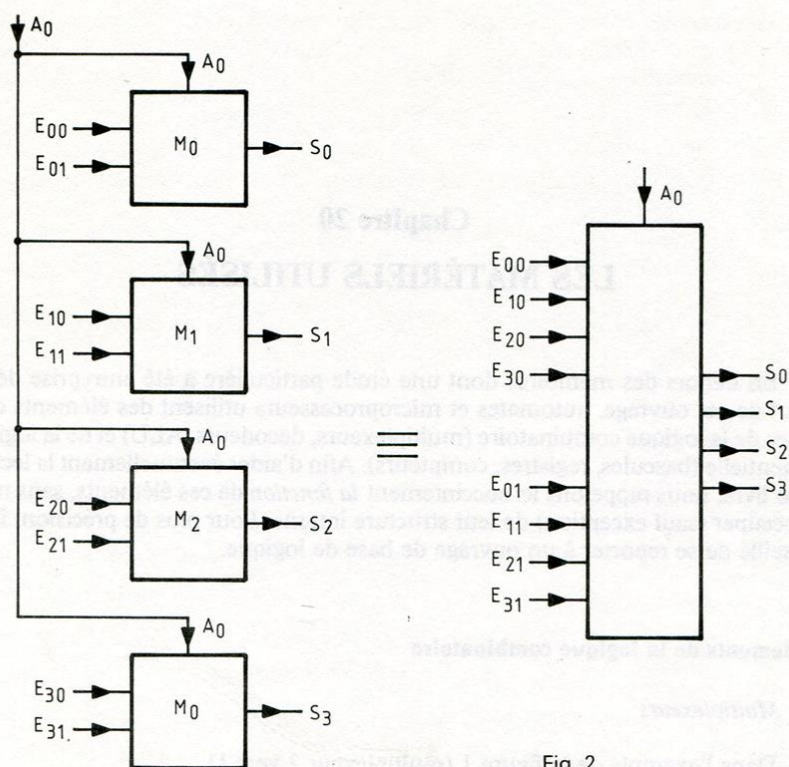


Fig. 2

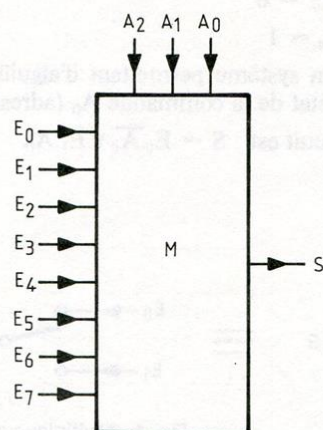


Fig. 3

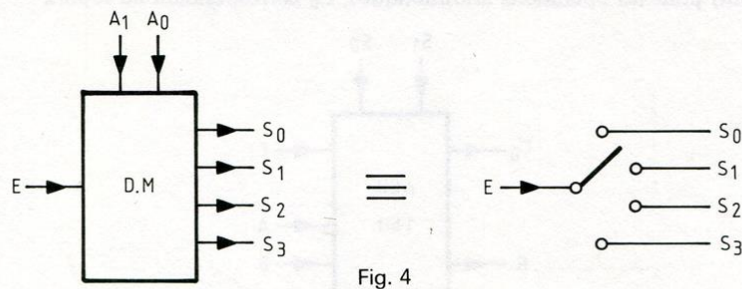
Il n'est pas difficile d'imaginer la possibilité d'association de tels multiplexeurs. La figure 2 représente un multiplexeur capable d'aiguiller vers la sortie (S_3, S_2, S_1, S_0) le nombre binaire ($E_{30}, E_{20}, E_{10}, E_{00}$) ou le nombre ($E_{31}, E_{21}, E_{11}, E_{01}$), selon l'état de la commande A_0 .

Extension : multiplexeur 8 vers 1 (fig. 3).

Pour être en mesure d'aiguiller 8 voies vers 1, il est nécessaire de disposer de 3 bits d'adresses ($2^3 = 8$).

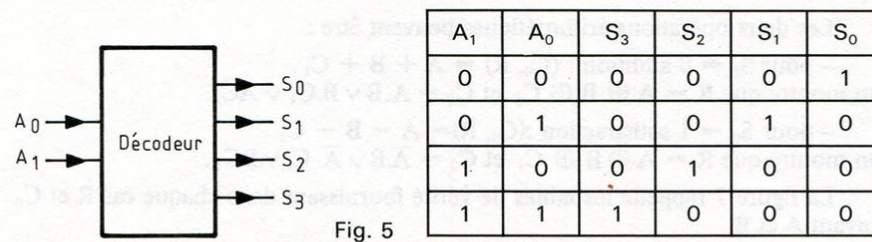
I. 2. Démultiplexeurs et décodeurs

Les *démultiplexeurs* réalisent la fonction inverse. La figure 4 donne l'exemple d'un démultiplexeur 1 parmi 4, nécessitant 2 bits d'adresses.



Suivant le code (A_1, A_0), l'une des quatre sorties S_0, S_1, S_2 ou S_3 est virtuellement reliée à l'entrée E. Les 3 sorties non sélectionnées sont, par exemple, maintenues à l'état 0.

Les *décodeurs* sont des démultiplexeurs dans lesquels l'entrée E est forcée à 1 (ou 0). L'état des entrées permet ainsi de reconnaître une des sorties en y imposant un état logique 1 (ou 0), les autres sorties demeurant à l'état 0 (ou 1). L'exemple d'un décodeur 1 parmi 4 est donné par la figure 5. Il est aisé de généraliser au décodeur 1 parmi 2^n qui permet la sélection d'une ligne parmi 2^n à partir de n variables binaires de décodage.



I. 3. Unité arithmétique et logique (ALU)

C'est l'organe de *traitement* des microprocesseurs. Une ALU permet d'effectuer sur deux nombres binaires présents sur ses entrées, certaines opérations logiques ou arithmétiques.

a) Exemple : ALU monobit complète, à deux bits de sélection (fig. 6).

C'est un bloc combinatoire permettant d'effectuer sur deux nombres de un bit A et B quatre opérations différentes, sélectionnées grâce aux quatre combinaisons possibles des deux bits de sélection S_1 et S_0 . Elle est complète dans le sens qu'elle permet, pour des opérations arithmétiques, de tenir compte d'un report (ou retenue) éventuel provenant d'une autre ALU, porté sur l'entrée C_i (carry in). Elle fournit le résultat sous forme d'un bit R pour les opérations logiques, de deux bits R et C_0 (carry out) pour les opérations arithmétiques, C_0 correspondant au report.

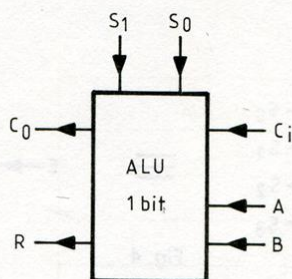


Fig. 6

Supposons que le bit S_1 soit utilisé pour la sélection de la nature de l'opération à effectuer : $S_1 = 0$, opération logique ; $S_1 = 1$, opération arithmétique.

Les deux opérations logiques peuvent être :

- pour $S_0 = 0$ « et » logique. Alors $R = A.B$ et $C_0 = 0$ pour $C_i = \emptyset$
- pour $S_0 = 1$ ou exclusif. Alors $R = A \oplus B = \overline{A}.B \vee A\overline{B}$ et $C_0 = 0$ pour $C_i = \emptyset$.

Les deux opérations arithmétiques peuvent être :

- pour $S_0 = 0$ addition : $(C_0, R) = A + B + C_i$.
On montre que $R = A \oplus B \oplus C_i$ et $C_0 = A.B \vee B.C_i \vee A.C_i$
- pour $S_0 = 1$ soustraction : $(C_0, R) = A - B - C_i$
On montre que $R = A \oplus B \oplus C_i$ et $C_0 = \overline{A}.B \vee \overline{A}.C_i \vee B.C_i$.

La figure 7 rappelle les tables de vérité fournissant dans chaque cas R et C_0 suivant A et B.

C_i	A	B	C_0	R
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1

$(S_1, S_0) = (0,0)$

Et logique

C_i	A	B	C_0	R
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$(S_1, S_0) = (1,0)$

Addition avec retenue

C_i	A	B	C_0	R	résultat en décimal
0	0	0	0	0	0
0	0	1	1	1	-1
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	1	1	-1
1	0	1	1	0	-2
1	1	0	0	0	0
1	1	1	1	1	-1

$(S_1, S_0) = (1,1)$

Soustraction avec retenue

C_i	A	B	C_0	R
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0

$(S_1, S_0) = (0,1)$

Ou exclusif

Fig. 7

Remarque : Dans le cas de la soustraction, le résultat est positif ou négatif. Lorsque le résultat est positif, $C_0 = 0$ et ce résultat est codé de façon naturelle. Lorsque le résultat est négatif, $C_0 = 1$ et ce résultat est codé en complément à 2. Par exemple le résultat $(C_0, R) = (1, 0)$ s'interprète par : nombre négatif (car $C_0 = 1$) de valeur absolue $(\bar{1}, 0) + 1 = (0, 1) + 1 = (1, 0)$ soit 2 en décimal.

b) Extension

A partir de n ALU à un bit complètes, il est possible de traiter des nombres de n bits. Pour les opérations arithmétiques, le résultat est un nombre de $n + 1$ bits. Le bit de poids le plus fort (MSB : most significant bit) est fourni par la sortie C_0 de l'ALU monobit de plus fort poids. Une ALU quatre bits, permettant quatre opérations (2 bits de sélection) est représentée sur la figure 8.

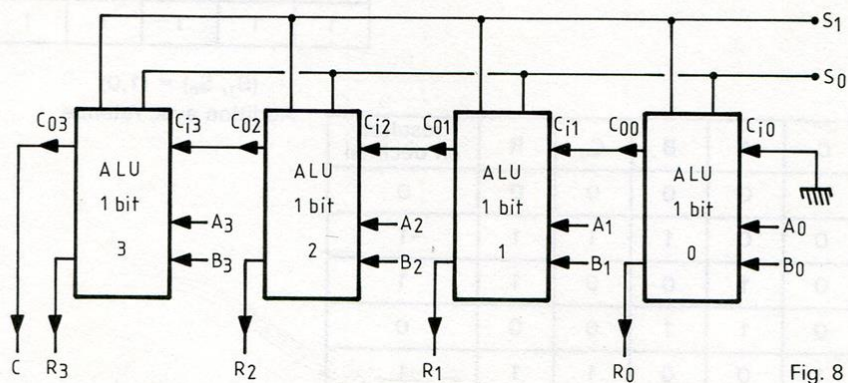


Fig. 8

c) Généralisation

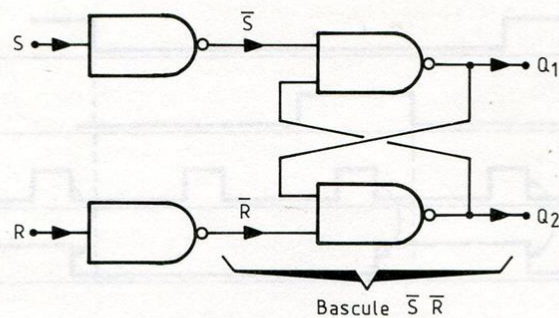
Il existe sur le marché des ALU intégrées qui permettent d'effectuer sur deux nombres binaires de format variable, de nombreuses opérations logiques et arithmétiques différentes, ce qui nécessite une augmentation du nombre des bits de sélection.

II. Eléments de la logique séquentielle

1) Bascule SR (fig. 9)

L'examen de la table de vérité montre que Q_2 et Q_1 sont complémentaires, sauf pour la combinaison (1,1) de (S,R). Nous nous interdisons cette combinaison et nous posons $Q_1 = Q$ et $Q_2 = \bar{Q}$.

– si $S = R = 0$, la sortie Q peut être à l'état 1 ou à l'état 0



S	R	Q ₁	Q ₂
0	0	0 ou 1	1 ou 0
0	1	0	1
1	0	1	0
1	1	1	1

Fig. 9

- si $S = 1$ (et $R = 0$), $Q = 1$
- si $R = 1$ (et $S = 0$), $Q = 0$
- si à partir de l'une des deux situations précédentes on revient à $S = R = 0$, Q ne change pas d'état : la bascule SR mémorise la dernière entrée R ou S qui a pris la valeur 1, S (set) mettant Q à 1, R (reset) mettant Q à 0.

Remarque : Une partie de cette bascule (bascule $\bar{S} \bar{R}$) est utilisée dans la maquette A comme circuit anti-rebonds associé au bouton poussoir.

2) Bascule SR synchrone (fig. 10)

La maîtrise des instants de commutation de la sortie Q est assurée par l'emploi d'une horloge de transfert Cl . Pour $Cl = 0$, la bascule ne prend pas en compte les états des entrées S et R . S et R se retrouvent en sortie des premières portes NAND dès que Cl passe à 1. La sortie Q se trouve donc synchronisée sur le front montant de Cl .

3) Bascule D (fig. 11)

C'est une bascule SR synchrone dans laquelle on impose la complémentation des entrées S et R . On pose $D = S = \bar{R}$. Ainsi, la sortie Q recopie l'entrée D au moment de la transition 0 à 1 de Cl et la mémorise à partir de cet instant.

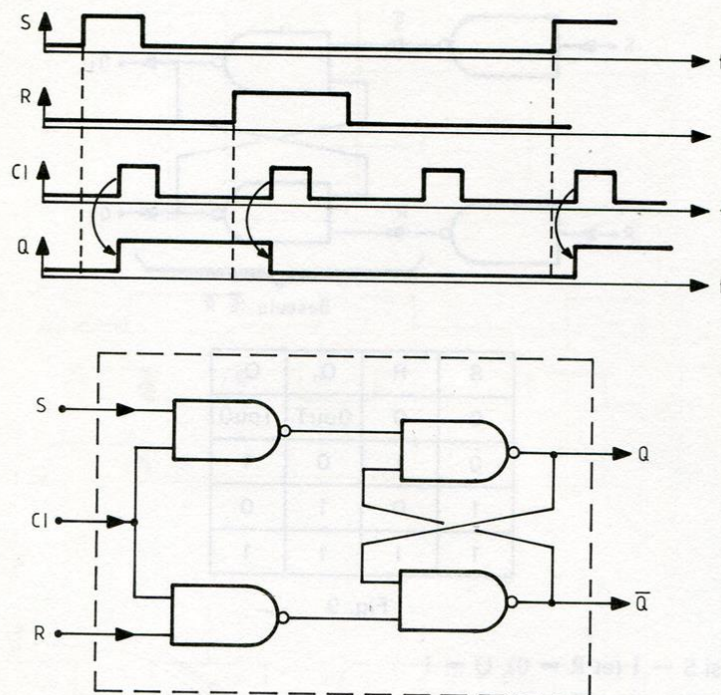


Fig. 10

Application : Registre D (fig. 12)

Un registre D de n bits est constitué par n bascules D commandées par la même horloge. Les n sorties recopient donc les n entrées au moment de la transition 0 à 1 de Cl et les mémorisent. En particulier, la transition 1 à 0 de Cl n'est sans aucun effet sur les sorties.

Remarque

Si dans ce montage, la sortie Q_i d'une bascule est reliée à l'entrée D_{i+1} de la bascule suivante (ou à l'entrée D_{i-1} de la bascule précédente), on obtient un registre à décalage à gauche (ou à droite) permettant l'entrée en série d'un mot.

4) Bascule JK (fig. 13)

Elle dérive de la bascule SR synchrone par deux bouclages supplémentaires qui interdisent la combinaison $S = R = 1$.

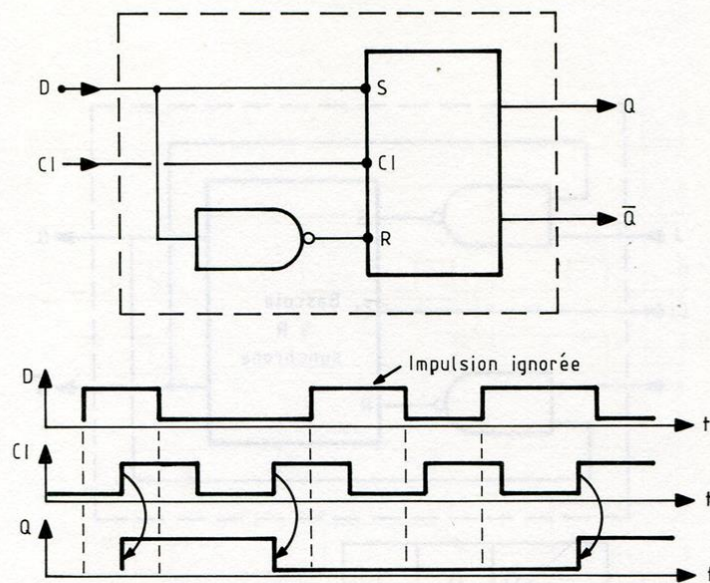


Fig. 11

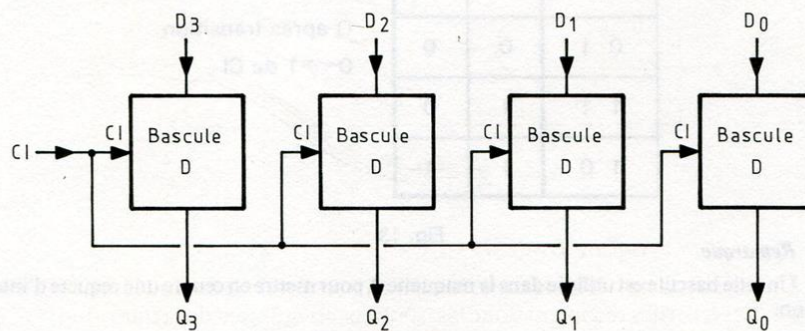
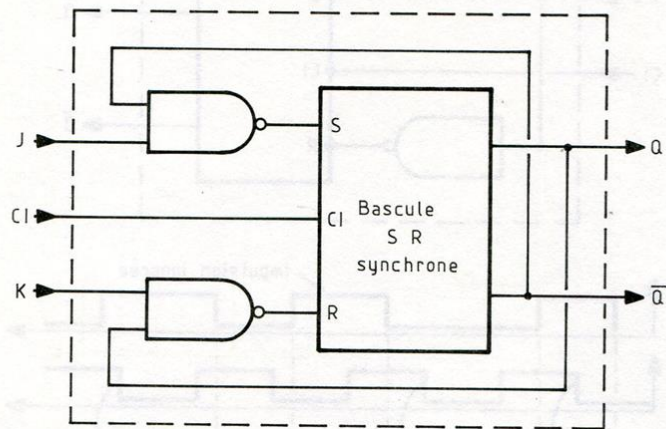


Fig. 12

Le tableau de la figure 13 indique le fonctionnement d'une telle bascule. C'est un tableau à trois entrées J, K et Q juste avant la transition 0 à 1 de Cl. Il donne l'état de la sortie Q juste après le front montant de Cl.

- si $J = K = 0$ Q n'est pas modifié
- si $J = K = 1$ Q change d'état au moment de la transition 0 à 1 de Cl
- si $J = 0$ et $K = 1$ Q est mis à 0 quel que soit son état précédent.
- si $J = 1$ et $K = 0$ Q est mis à 1 quel que soit son état précédent.



J \ K Q	0	1
0 0	0	1
0 1	0	0
1 1	1	0
1 0	1	1

Q après transition
 0 → 1 de Cl

Fig. 13

Remarque

Un telle bascule est utilisée dans la maquette A pour mettre en œuvre une requête d'interruption.

Application : bascule Cl ou bascule T (fig. 14).

C'est une bascule JK pour laquelle $J = K = 1$. Elle permet une division par deux de la fréquence du signal d'horloge.

Application de la bascule T : compteur binaire (fig. 15).

Il peut résulter de l'association en cascade de bascules T.

La figure 16 représente le chronogramme de fonctionnement. Les codes (S_2, S_1, S_0) lus successivement sont la représentation en binaire des nombres de 0 à 7. Ce compteur présente un grave défaut d'incohérence dû à son caractère asynchrone, qu'il ne nous appartient pas de développer ici,

Le contenu du compteur (S_2, S_1, S_0) augmente d'une unité au moment du front montant de Cl.

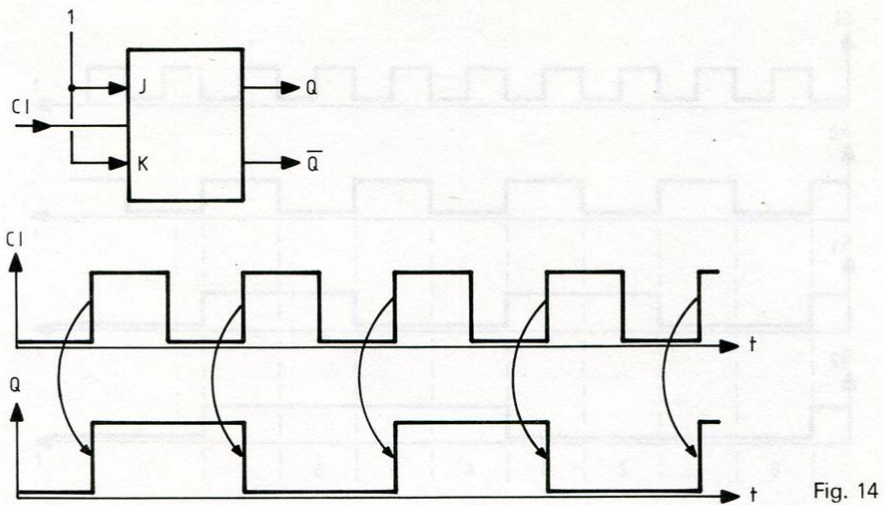


Fig. 14

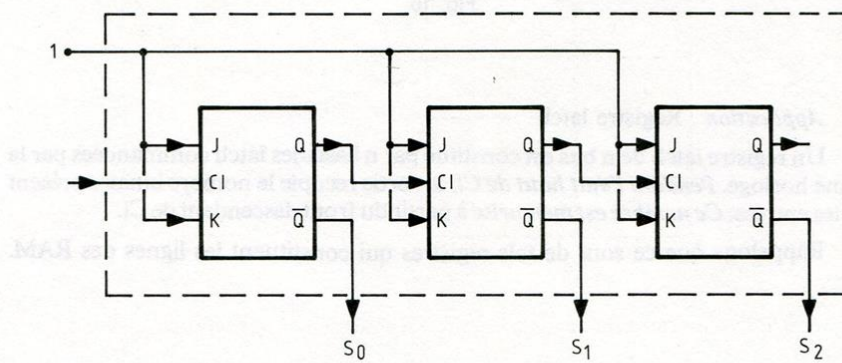


Fig. 15

5) Bascule latch (fig. 17)

Cette bascule est souvent confondue avec la bascule D, son fonctionnement est pourtant différent et il est essentiel de bien l'assimiler.

Si $CI = 1$, $A = E$ et $B = 0$ alors $Q = A = E$. La bascule est dite transparente : sa sortie recopie son entrée.

Si $CI = 0$, $A = 0$ et $B = Q$. Les deux états $Q = 0$ et $Q = 1$ sont possibles. La bascule latch mémorise l'état dans lequel se trouve l'entrée au moment du front descendant de CI.

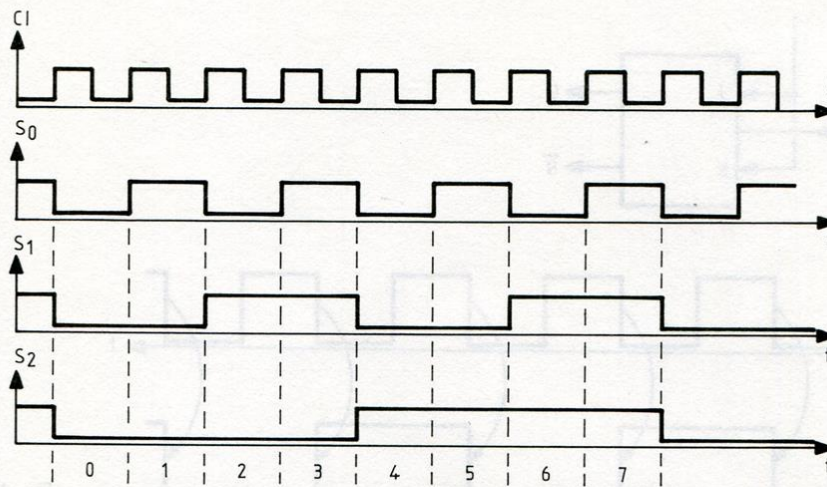


Fig. 16

Application : Registre latch

Un registre latch de n bits est constitué par n bascules latch commandées par la même horloge. Pendant l'état haut de Cl , la sortie recopie le nombre binaire présent sur les entrées. Ce nombre est mémorisé à partir du front descendant de Cl .

Rappelons que ce sont de tels registres qui constituent les lignes des RAM.

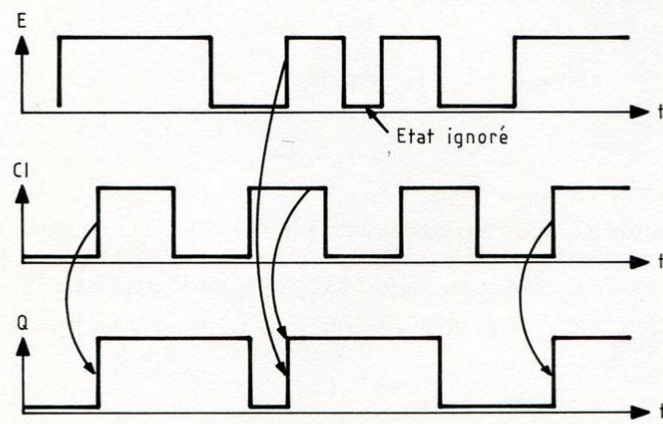
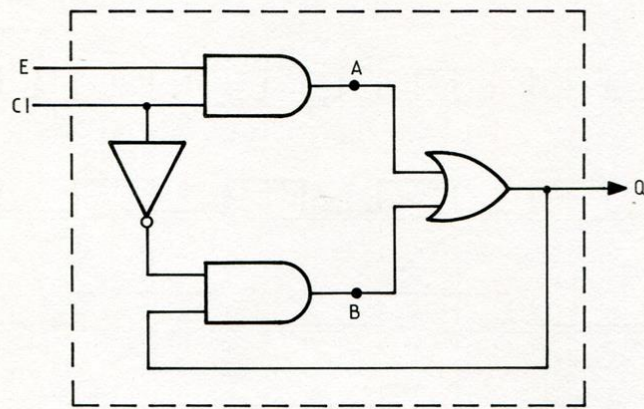


Fig. 17

REMERCIEMENTS

Les auteurs tiennent à remercier :

M. Jean-Marc Lepetit, professeur au lycée Dorian, pour son importante contribution à la mise au point de la maquette.

M. Jourdan (R.E.A.) pour ses précieux conseils lors de la mise en œuvre du microprocesseur CDP 1802.

BIBLIOGRAPHIE

Les systèmes microprogrammés, par Maurin et Robin, Dunod.

Data Book « COS/MOS » Integrated circuits », RCA.

User Manual for the CDP 1802 COSMAC Microprocessor », RCA.

UN MICROPROCESSEUR PAS A PAS

Ce nouvel ouvrage tire pour une grande part son originalité de son caractère résolument pédagogique :

Ses auteurs, deux professeurs électroniciens, y proposent au technicien de l'industrie, à l'étudiant ou à l'amateur intéressé, **une formation très progressive** au microprocesseur. Il est invité à utiliser **une maquette** facile à réaliser qui le place immédiatement **sur le terrain expérimental**. L'exposé est d'ailleurs toujours mêlé d'applications entièrement développées que l'on peut soi-même étendre, comme le montre le sommaire.

Principaux chapitres :

- Les mémoires.
- Automate programmable simple et composé.
- Notion de processeur.
- Structure du microprocesseur.
- Les instructions du COSMAC CDP 1802.
- Conception d'une maquette d'étude.
- Réalisation pratique des maquettes A et B.
- Etude en pas à pas d'un programme élémentaire.
- Branchements inconditionnel et conditionnel.
- Sous-programmes.
- Entrée et sortie.
- Interrupteur.
- Introduction des données.
- Affichage numérique.
- Conversion numérique \Leftrightarrow analogique.

