Initiation à la programmation des microprocesseurs

par J.-C. TRIGEASSOU,
L.T.E. Louis-Armand, 86000 Poitiers.

I. INTRODUCTION.

L'exposé suivant relate deux années d'expérience (1977-1978 et 1978-1979) de l'enseignement de la micro-informatique en TF_5, classe préparant au baccalauréat de technicien en Physique (micro-informatique = informatique des microprocesseurs).

Pourquoi ce cours nouveau ? Les élèves de la série F_5 sont initiés à la mesure physique et il a paru souhaitable de leur donner des notions d'informatique et de micro-informatique en raison de l'automatisation de la fonction mesure. Ces notions ne figurent pas au programme actuel de cette classe, ni celles relatives à l'algèbre de Boole, indispensables pour aborder le microprocesseur. Un horaire a été aménagé, sous la forme d'un cours semestriel de deux heures hebdomadaires par élève. L'enseignement est du type T.P. - cours, avec la participation la plus étroite possible de l'élève, compte tenu des moyens matériels.

Ce cours est en fait une initiation, les exemples choisis sont volontairement élémentaires, leur progression permettant une découverte « naturelle » des instructions du microprocesseur. Il peut aussi servir de canevas pour une approche du microprocesseur en autodidacte. Le matériel indispensable consiste en un simulateur ou kit d'initiation (avec le microprocesseur : SCMP). Les modules périphériques sont très simples et facilement réalisables.

L'idée directrice de cet enseignement de micro-informatique a été la suivante : présenter le simulateur (kit d'initiation) comme une calculatrice programmable, la différence essentielle résidant au niveau du langage (langage relativement évolué, mathématique et décimal pour la calculatrice, binaire ou hexadécimal, composé de fonctions logiques et arithmétiques les plus simples pour le microprocesseur). En conséquence, les élèves sont d'abord initiés à la programmation des calculatrices, ce qui permet aussi de démystifier le terme d'informatique. Ensuite sont introduites les notions indispensables de numération et d'arithmétique binaires. Il est alors possible d'aborder le micro-

processeur avec des exercices de simple arithmétique. Puis on élargit le champ d'application aux possibilités d'entrées-sorties, nouveauté par rapport à la calculatrice.

Signalons par ailleurs qu'en raison de la technologie binaire du microprocesseur, on pouvait s'attendre à une longue initiation aux circuits logiques et à l'algèbre de Boole avant d'aborder la programmation. En réalité, l'expérience montre que bien qu'indispensables, seuls quelques rudiments (numération binaire, fonctions de base) sont nécessaires à l'étude de la logique programmée. Par contre, si on désire réaliser un système complet avec un microprocesseur (étude du matériel et du logiciel), il faut posséder de solides notions en circuits logiques. Mais la première démarche est suffisante dans bon nombre d'applications et particulièrement pour l'utilisation des automates programmables, très voisins des kits d'initiation.

II. LES CALCULATRICES PROGRAMMABLES.

Cette première partie est la plus ancienne du cours. Dès l'apparition des calculatrices programmables, les élèves de F_5 furent initiés à l'informatique. Le matériel retenu avait été à l'époque le modèle HP 25 de Hewlett-Packard. Actuellement, on peut se procurer à des prix plus faibles (environ 300 F) des appareils plus sophistiqués (sous-programmes, etc). On peut aussi utiliser de véritables ordinateurs en langage évolué (BASIC) pour moins de 10 000 F. L'intérêt des calculatrices programmables est lié à leur faible prix : plusieurs postes de manipulation pour un investissement réduit. Un autre avantage de ces calculatrices est que leur « langage » de programmation est simple et souple d'utilisation, tout en étant relativement voisin de celui des microprocesseurs.

L'enseignement est composé de trois séances de deux heures. La première séance est consacrée à la découverte de l'appareil : apprentissage du calcul en logique polonaise inversée (particulier à la marque HP mais non fondamental en informatique) et premier essai de programmation. L'exemple choisi est le calcul d'un polynôme $P(x) = ax^2 + bx + c$. Le calcul est effectué manuellement, puis toujours manuellement en affectant des mémoires aux coefficients a, b et c et à la variable x (découverte des instructions STO et RCL et des mémoires de données). Ensuite, on passe naturellement à l'écriture du programme : enregistrement en mémoire programme de l'ordre d'appui sur les touches. De ce premier essai, l'élève doit retirer les notions de mémoire (de donnée et de programme) et de séquence (calcul effectué par étape, dans l'ordre indiqué).

La deuxième séance est plus ambitieuse : à l'occasion du calcul de e^x, on cherche à dégager les notions d'algorithme et d'organigramme. e^x est calculé par le développement en série :

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Les élèves proposent alors de calculer e^x comme un polynôme et ils butent immédiatement sur la longueur des calculs. On peut ainsi dégager l'intérêt d'une méthode plus astucieuse et plus économique : l'algorithme. Ainsi,

$$e^x{}_n = R_0 + R_1 + \ldots + R_n$$

avec :

$$e^x{}_n = e^x{}_{n-1} + R_n \quad \text{et} \quad R_n = \frac{x}{n} R_{n-1}$$

($e^x{}_n$: approximation au rang n de e^x).

Il est nécessaire d'introduire un test de précision afin d'arrêter le processus de calcul : arrêt si $|R_n| < \varepsilon$ (ε : précision). Cet algorithme établi, on cherche une représentation commode des opérations à effectuer : l'organigramme (voir fig. 1). On apprend à lire cet organigramme, en dégageant la notion de boucle de calcul, liée à un saut conditionnel ou non.

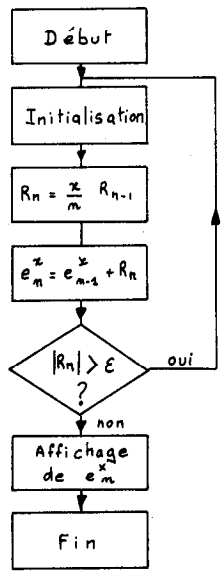


Fig. 1

La troisième séance est un exercice d'application pour l'élève, qui sert de test sur l'acquisition des connaissances précédentes : par exemple le problème de la charge d'un condensateur, ce qui revient à intégrer une équation différentielle du 1er ordre. L'algorithme est défini en commun, puis les élèves établissent eux-mêmes l'organigramme et sa traduction en langage machine. L'exploitation de ce programme est simple, il permet en outre de comparer les résultats théoriques obtenus avec les courbes expérimentales tracées en manipulation d'électricité.

III. L'ALGEBRE DE BOOLE.

Actuellement, cette partie de l'électronique n'est pas au programme de la section F_5. Disposant de deux séances de deux heures, il a donc fallu ne retenir que l'essentiel pour la compréhension du microprocesseur.

La première séance est consacrée à la numération binaire et aux fonctions logiques. Un parallèle est présenté entre numération décimale et binaire, cette dernière étant justifiée par des raisons de technologie et de fiabilité pour l'ordinateur. Puis sont exposés les différentes techniques de conversions (décimal binaire et binaire décimal), les lois de l'addition binaire, le système hexadécimal et le système BCD (binaire codé décimal). Une autre partie est consacrée aux fonctions de base de la logique : fonction inversion, fonction OU et fonction NOR, fonction ET et fonction NAND, fonction OU exclusif. Pour chaque fonction sont présentés la table de vérité, la formule algébrique et le symbolisme. Des vérifications expérimentales sont effectuées sur un simulateur logique réalisé avec des circuits intégrés TTL : fonctions NAND et NOR.

La deuxième séance est consacrée à deux éléments essentiels d'un ensemble de calcul binaire : l'additionneur et la mémoire élémentaire. En rappelant le principe de l'addition, on établit la table de vérité dans le cas de deux bits *a* et *b*, ce qui conduit au schéma logique du demi-additionneur (fig. 2). Ensuite, on passe à l'additionneur complet, tenant compte de la retenue de rang inférieur. Les élèves ne connaissent pas les techniques de simplification logique, on arrive néanmoins au schéma de la fig. 3 grâce à des variables logiques intermédiaires.

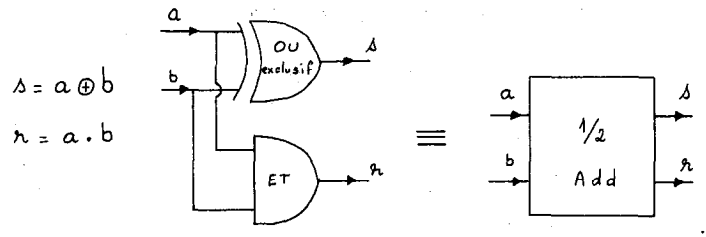


Fig. 2

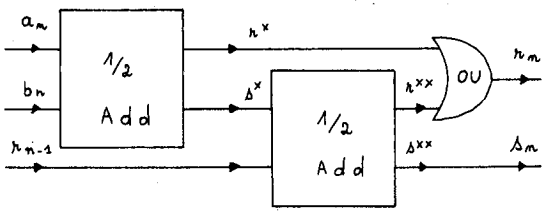


Fig. 3

L'autre partie est consacrée à la mémorisation des données binaires. On débute par la bascule RS réalisée avec des circuits NOR, ce qui permet d'introduire la bascule D simple puis la bascule D avec entrée d'autorisation T (ou entrée d'horloge) (fig. 4).

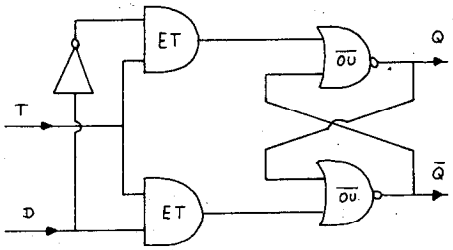


Fig. 4

Cette bascule qui permet de mémoriser un bit est présentée comme la mémoire élémentaire. Le simulateur logique est évidemment mis à profit pour vérifier tous les schémas logiques proposés.

L'additionneur et l'élément de mémoire permettent enfin de présenter et de justifier, de manière très simplifiée, l'organisation d'un ordinateur ou d'un microprocesseur. Il est alors possible d'aborder la micro-informatique.

IV. PROGRAMMATION D'UN MICROPROCESSEUR : CAS DU SC. MP.

Pour alléger l'écriture, on utilisera l'abrévaition µP pour microprocesseur.

1. Introduction.

Les exercices qui suivent sont réalisables sur le µP SCMP (pour un autre µP, ils serviront néanmoins de guide). Il faut

pour cela disposer d'un simulateur ou kit d'initiation. Dans le cas présent, le modèle employé a été le système EMR (voir en annexe la description du μP et du simulateur).

Le fait de ne disposer que d'un appareil, de dimension réduite, a été un écueil : il a été surmonté par l'emploi d'un circuit fermé de télévision (la caméra doit posséder une optique appropriée pour ne viser que l'ensemble clavier-afficheur).

Signalons que pour suivre cette partie de l'exposé, il est indispensable de posséder un minimum de connaissances en logique binaire (voir en annexe).

Après une présentation sommaire du μP utilisé et quelques définitions (accumulateur, données, octet, case mémoire, etc.), on passe directement à un premier essai de programmation. On exploite pour cette première approche la similitude entre calculatrice programmable et simulateur à μP, similitude renforcée par la présentation identique du clavier et de l'afficheur. Il eut pu paraître souhaitable de présenter le tableau général des instructions, mais leur caractère de nouveauté risquait de décourager les élèves. Le choix a été fait d'une découverte progressive, à l'occasion des exercices.

Remarque.

Pour alléger l'écriture, on emploie uniquement la numération hexadécimale, sauf exception.

2. **Premier exercice : addition.**

Soit à additionner deux nombres A et B de 8 bits chacun (un octet) S = A + B, par exemple :

$$(46)_H + (24)_H = (6A)_H.$$

Rappelons d'abord le principe de l'opération en binaire :

$$\begin{array}{r} (0100\quad 0110)_B \\ +\,(0010\quad 0100)_B \\ \hline (0110\quad 1010)_B \end{array} \equiv \begin{array}{r} (46)_H \\ +\,(24)_H \\ \hline (6\,A)_H \end{array}.$$

La méthode employée par le μP pour effectuer cette opération peut paraître, à première vue, déroutante. Ainsi toutes les instructions exécutées sont référencées par rapport à l'accumulateur (AC) :

1) on place (charge) le nombre A dans l'accumulateur,
2) on additionne B à l'accumulateur, soit B + AC = B + A = S,
3) pour terminer, on range le contenu de l'accumulateur (S) dans une case mémoire M où il pourra être lu après l'exécution du programme.

Ecrivons l'organigramme correspondant (fig. 5) :

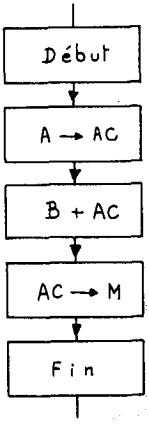


Fig. 5

Une fois acceptée cette méthode de calcul, voyons comment elle est effectivement réalisée.

1) A → AC : on utilise une instruction immédiate, en deux octets LDI, (A)

1er octet : LDI : quand le μP rencontre LDI, il sait qu'il doit placer immédiatement (charger : to load) dans l'accumulateur le contenu de l'octet suivant, ici (A).

2) B + AC : c'est encore une instruction immédiate qui opère selon le processus précédent. ADI, (B) : le μP additionne en binaire à l'accumulateur l'octet suivant, soit (B).

3) Cette opération effectuée, l'accumulateur contient (A + B) = (S), on doit ranger ce résultat dans une case mémoire M (sinon il serait détruit). Pour cela, on utilise une instruction du type ST, de to store : ranger. Elle ressemble à l'instruction STO des calculatrices, mais elle en diffère par son mode d'adressage. Dans une calculatrice, il suffit de préciser le numéro de la mémoire. Dans le cas du μP, on fait un rangement par rapport au compteur ordinal (PC), par exemple une case plus loin. C'est un adressage de type indexé : on atteint une case mémoire au moyen d'un déplacement ici $(+1)_D$ (une case plus loin) par rapport à un index : ici le compteur ordinal (on peut aussi utiliser des pointeurs comme on le verra plus loin).

L'instruction de rangement s'écrit :

$$\text{ST/PC, } (01)_H \quad \text{car} \quad (+1)_D = (01)_H.$$

Le programme définitif s'écrit :

$A \rightarrow AC$	LDI, (A)
$B + AC$	ADI, (B)
$AC \rightarrow M$	ST/PC, $(01)_H$.

Il faut à présent introduire ce programme en mémoire vive (RAM) pour que le μP puisse l'exécuter. On se pose alors la question : comment peut-on écrire : LDI, (A) à partir du clavier ? Cela est possible directement sur des simulateurs évolués avec langage assembleur (ce sont de véritables ordinateurs). En fait sur notre simulateur, on ne peut employer que l'écriture hexadécimale. En conséquence, on écrit le code hexadécimal de chaque instruction, à raison d'un octet par case mémoire. Par exemple, LDI, A est traduit par $(C4)_H$, $(46)_H$ (avec $A = (46)_H$. On se pose aussi la question : à quel endroit (à quelle adresse) de la mémoire vive (RAM) peut-on écrire le programme ? Supposons une version de base de simulateur dont la mémoire vive est située de 0F00 à 0FFF, alors on peut écrire le programme à partir de 0F00 (par exemple), soit :

Adresse	Code mnémonique	Code hexadécimal
0F00	LDI	C4
0F01	46	46
0F02	ADI	F4
0F03	24	24
0F04	ST/PC	C8
0F05	01	01
0F06	— — —	— — ← M

Précisons à présent le mécanisme de l'instruction ST. Le μP décrit séquentiellement le programme à partir de 0F00. Lorsqu'il rencontre l'instruction ST/PC, $(01)_H$ le compteur ordinal (PC) a la valeur 0F05. Alors le μP range le contenu de l'accumulateur dans la case mémoire M dont l'adresse est :

$$(0F05)_H + (01)_H = (0F06)_H \qquad (01)_H = \text{déplacement.}$$

Donc M est la case mémoire d'adresse $(0F06)_H$, case où pourra être lu le résultat.

Remarquons dès à présent que le déplacement peut être positif ou négatif : c'est-à-dire après ou avant l'index (ici PC). Par ailleurs, comme ce déplacement est codé en binaire algébrique sur un octet (voir annexe), il peut valoir au maximum $(+ 127)_D$ en positif ou $(— 128)_D$ en négatif.

Pour terminer, on introduit le programme en mémoire, on pointe la case mémoire de début et on lance l'exécution. Pre-

mière déception, on ne voit pas le résultat sur l'afficheur, au contraire, il clignote ou indique des caractères bizarres. Pour lire le résultat, on doit arrêter l'exécution en appuyant sur la touche de RAZ, former l'adresse de M (soit 0F06) et lire enfin le résultat (soit $(6A)_H$).

Précisons d'autre part qu'il n'y a aucune raison pour que le μP s'arrête lui-même : il débute l'exécution du programme en 0F00 et arrive en 0F06 en quelques dizaines de μs, il poursuit son chemin, au besoin en détruisant ce qui est écrit en M. On peut, si on le désire, provoquer une remise à zéro (RAZ) du compteur ordinal à l'adresse $(0F07)_H$. Il suffit pour cela d'utiliser l'instruction XPPC/3 = $(3\,F)_H$. Ce code n'a rien de mystérieux ; il signifie que l'on échange le compteur ordinal avec le pointeur 3 (qui contient toujours $(0000)_H$ sauf si on y a écrit une autre valeur). Cette instruction exécutée, le compteur ordinal a pour nouvelle valeur $(0000)_H$, il s'incrémente et passe à $(0001)_H$, début du programme de gestion : on a le même résultat que si on avait appuyé sur la touche RAZ.

On peut effectuer des variantes de programmation : changer les nombres, vérifier qu'il peut se produire un dépassement, ranger le résultat dans une autre case M, faire une addition décimale ou BCD au moyen de l'instruction DAI (addition décimale immédiate, avec les valeurs précédentes, on doit obtenir comme résultat $(70)_H$).

3. Deuxième exercice : addition améliorée.

Dans l'exercice précédent, un des plus simples que l'on puisse réaliser à l'intention de débutants, les données (A, B et S) sont mélangées avec des instructions (LDI, ADI, ST). On va à présent refaire la même addition, mais en séparant la mémoire en zone programme et zone données. Par exemple, le programme sera encore écrit à partir de $(0F00)_H$, mais les données seront situées à partir de $(0FA0)_H$. En raison de cette séparation, on est conduit à utiliser un pointeur, par exemple P1, qui sert d'index. On a besoin de trois cases mémoires pour A, B et S, soit :

0FA0 = M1 contient A,
0FA1 = M2 contient B,
0FA2 = M3 contient S.

P1 sera chargé avec l'adresse de la première case, soit $(0FA0)_H$.

Remarque.

On ne doit pas confondre P1 qui mémorise une adresse (ici $(0FA0)_H$ avec le contenu de la case mémoire dont P1 indique l'adresse (ici M1 = A).

Le programme est semblable au précédent, la première partie consiste à charger P1 avec l'adresse $(0FA0)_H$, la deuxième partie étant l'addition.

— P1 = $(0FA0)_H$.

P1 est un registre d'adresse composé de deux octets (partie haute et partie basse) :

$P_1 = (P_{1H})(P_{1L})$ P_{1H} : partie haute (H pour High)

P_{1L} : partie basse (L pour Low)

dans notre exemple : P1 = 0FA0, soit :

$P_{1H} = (0F)_H$ et $P_{1L} = (A0)_H$.

On charge une partie de pointeur en échangeant le contenu de l'accumulateur avec P_{iH} ou P_{iL} grâce aux instructions XPAH/i et XPAL/i (avec i = 1, 2, 3), ce qui conduit à l'organigramme suivant (fig. 6) :

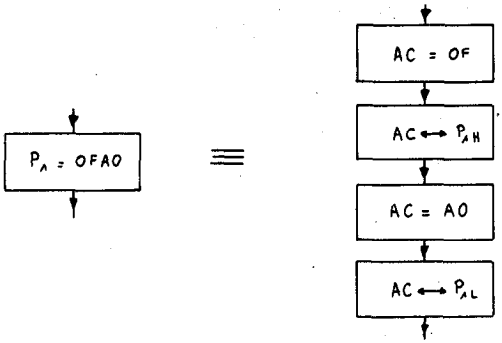


Fig. 6

soit : AC = $(0F)_H$ = LDI, 0F

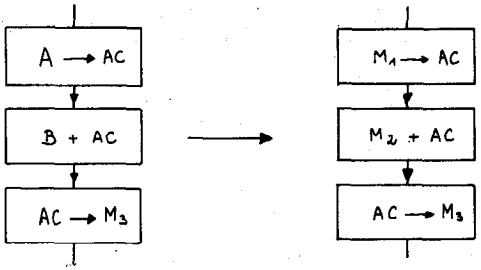
AC ⟷ P_{1H} = XPAH/1

AC = $(A0)_H$ = LDI, A0

AC ⟷ P_{1L} = XPAL/1

— S = A + B

l'organigramme précédent devient (fig. 7) :



Commentaires.

— $M_1 \rightarrow AC$: on charge A dans l'accumulateur, mais comme A est situé en 0FA0, on utilise une instruction de charge-

ment (LD) indexée par rapport à P1, en précisant le déplacement (octet suivant) par rapport à l'index, en positif ou en négatif (dans ce cas déplacement : $(0)_D = (00)_H$ car P1 contient l'adresse de M1),

donc LD/1, $(00)_H$.

— M2 + AC :

Même procédé, on additionne le contenu de M2 au contenu de l'accumulateur, grâce à une instruction ADD indexée par rapport à P1, en précisant une case plus loin car :

$$\underbrace{(0FA0)_H}_{P_1} + \underbrace{(01)_H}_{\text{déplacement}} = \underbrace{(0FA1)_H}_{\text{adresse de } M_2}$$

donc ADD/1, $(01)_H$.

— AC →M3 :

C'est une instruction de rangement indexée par rapport à P1 (au lieu de PC précédemment) avec un déplacement égal à $(02)_H$ pour atteindre M3,

donc ST/1, $(02)_H$.

— programme définitif :

Adresse	Code Mnémonique	Code hexadécimal
0F00	LDI	C4
0F01	0F	0F
0F02	XPAH/1	35
0F03	LDI	C4
0F04	A0	A0
0F05	XPAL/1	31
0F06	LD/1	C1
0F07	00	00
0F08	ADD/1	F1
0F09	01	01
0F0A	ST/1	C3
0F0B	02	02

Remarque.

Ne pas oublier de placer les nombres A et B dans les cases M1 et M2, lire le résultat en M3.

Variantes : comme précédemment.

4. Troisième exercice : allumer une lampe par l'action d'un programme.

Cet exercice est composé de trois parties de complexité croissante, dont le but est de montrer qu'on peut travailler en relation avec le milieu extérieur. On met à profit l'existence sur le boîtier μP de deux entrées logiques (S_A et S_B) et de trois sorties

logiques (F0, F1 et F2) correspondant à des cases binaires du registre d'état (SR) suivant le schéma de la fig. 8.

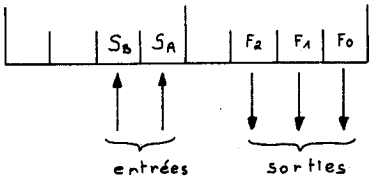


Fig. 8

a) LE PROGRAMME PERMET D'ALLUMER UNE LAMPE.

Dans ce cas, on utilise la sortie F0 (ou F1 ou F2) chaque sortie (ou flag) est en fait mémorisée par une bascule binaire. Donc, si F0 = 0, la sortie est au niveau 0 volt, si F0 = 1, elle est au niveau + 5 volts, de manière permanente dans chaque cas. De plus, chaque sortie supporte une charge TTL, c'est-à-dire que l'on peut brancher au maximum une entrée de circuit logique TTL par sortie. Si on désire allumer une petite lampe ou une diode luminescente, on peut utiliser un des schémas de la fig. 9.

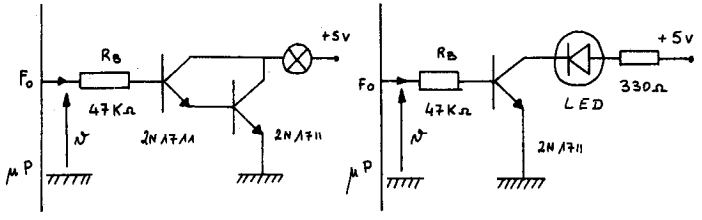


Fig. 9

La lampe (ou la diode) s'allume lorsque $v = 5$ V, soit pour F0 = 1.

Remarque.

Lorsqu'on fait une remise à zéro (touche RAZ), tous les registres du μP sont remis à zéro, en particulier SR ainsi que ses sorties Fi. Donc pour que Fi prenne la valeur 1, il faut l'imposer par programme.

Supposons dans notre cas que F0 = 1 et que pour simplifier, toutes les autres cases soient à zéro (même pour les entrées). Donc : SR : $(0\,000\,0001)_B$

soit en hexadécimal SR : $(01)_H$.

On utilise l'instruction CAS qui permet de copier le contenu de l'accumulateur dans le registre d'état avec l'organigramme de la fig. 10.

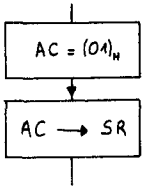


Fig. 10

Commentaires.

— AC = $(01)_H$ on écrit $(01)_H$ dans l'accumulateur soit LDI, $(01)_H$

— AC → SR on utilise l'instruction CAS.

Programme.

Adresse	Code mnémonique	Code hexadécimal
0F00	LDI	C4
0F01	01	01
0F02	CAS	07

Exploitation.

Après lancement du programme, la lampe s'allume. On l'éteint en appuyant sur la touche RAZ.

b) La lampe s'allume lorsque SA = 1.

On met à profit la possibilité d'entrée logique testable par programme. On utilise pour cela l'entrée SA (ou SB). La commande est matérialisée par un interrupteur K selon le schéma de la fig. 11.

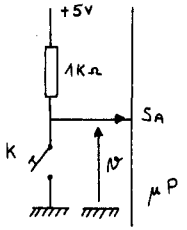


Fig. 11

si K fermé $v = 0$ donc SA = 0,

si K ouvert $v = +5$ V donc SA = 1.

Problème.

On teste l'entrée SA.

Si SA = 1, on allume la lampe (soit F0 = 1) ce qui correspond à l'organigramme de la fig. 12.

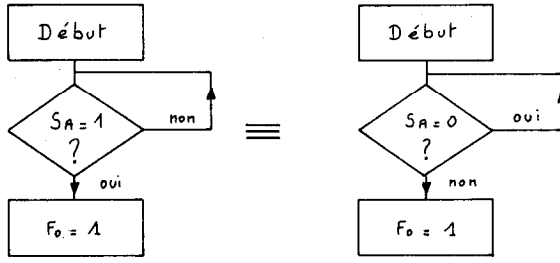


Fig. 12

On utilise la deuxième forme du test car pratiquement, on ne peut faire un saut que pour une réponse affirmative.

SA = 0 ? scrutation et test de SA.

Pour commencer, on recopie le registre d'état dans l'accumulateur par l'instruction complémentaire de CAS : soit CSA : SR → AC.

Deux cas peuvent se présenter :

SA = 0 (xxx 0 xxxx) SA = 1 (xxx 1 xxxx).

Les valeurs des autres cases binaires nous sont ici indifférentes, pour cela on doit isoler le bit significatif de SA : on y parvient par un ET logique immédiat (ANI) du contenu de l'accumulateur avec l'octet $(10)_H = (0001\ 0000)_B$, le résultat du ET logique apparaît ensuite dans l'accumulateur.

Envisageons les deux possibilités :

1) SA = 0 xxx 0 xxxx
(10) 000 1 0000
(AC) . (10) = 000 0 0000 = $(00)_H$

2) SA = 1 xxx 1 xxxx
(10) 000 1 0000
(AC) . (10) = 000 1 0000 = $(10)_H$

On remarque donc que l'accumulateur contient $(00)_H$ si SA = 0 et qu'il est différent de $(00)_H$ si SA = 1. Pour le test (saut conditionnel), on a le choix entre JZ (saut si AC = $(00)_H$ JNZ (saut si AC $\neq$ $(00)_H$) et JP (saut si AC > 0). Dans notre cas, on utilise JZ.

L'octet suivant l'instruction de saut (conditionnel ou non) indique le déplacement $(XY)_H$ (algébrique) qu'il faut effectuer par rapport à la valeur du compteur ordinal (ou d'un pointeur) pour aboutir à l'endroit désiré. On déterminera $(XY)_H$ par la suite.

Si SA $\neq$ 0, on poursuit normalement le programme qui est constitué par le segment du paragraphe *a*).

Ecrivons le programme définitif en le faisant précéder par l'instruction NOP (pas d'instruction) à cause du saut.

Adresse	Code mnémonique	code hexadécimal
0F00	NOP	08
0F01	CSA	06
0F02	ANI	D4
0F03	10	10
0F04	JZ	98
0F05	$(XY)_H$	FB
0F06	LDI	C4
0F07	01	01
0F08	CAS	07

Calcul de $(XY)_H$.

Depuis l'adresse $(0F05)_H$ on doit revenir au début, c'est-à-dire à l'adresse effective $(0F01)_H$. Avec le SCMP, on doit procéder de la manière suivante : on revient une case avant l'adresse effective, soit ici en $(0F00)_H$, cette case n'est pas prise en compte par le μP qui incrémente son compteur ordinal et redémarre le traitement en $(0F01)_H$.

$(XY)_H$ est une quantité algébrique qui, additionnée à la valeur prise par le compteur ordinal (ou un pointeur), indique l'adresse de retour,

ici $(0F05)_H + (XY)_H = (0F00)_H$,

donc $(XY)_H = (-5)_D = (FB)_H$. (voir annexe).

Remarque.

Comme on doit revenir momentanément en $(0F00)_H$, on remplit cette case mémoire avec une instruction sans effet = NOP.

Exploitation du programme.

— S'assurer au départ que SA = 0.

— Lancer le programme, la lampe reste éteinte.

— Faire SA = 1, la lampe s'allume.

Pour SA = 0, la lampe ne s'éteint pas, car cela n'a pas été encore prévu, on l'éteint par la touche RAZ.

c) LA LAMPE EST ALLUMÉE SI SA = 1, ÉTEINTE SI SA = 0.

On complète le programme précédent en ajoutant la condition lampe éteinte si SA = 0. Il est à remarquer qu'à présent on doit pouvoir éteindre la lampe après l'avoir allumée (F0 = 0), ce qui conduit à l'organigramme de la fig. 13.

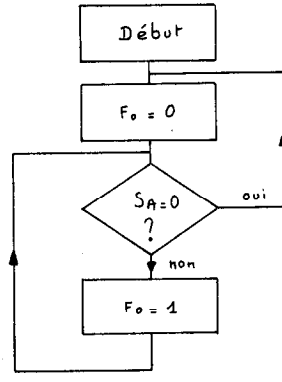


Fig. 13

On obtient ainsi le premier programme entièrement bouclé (à deux boucles). Il est constitué de segments des programmes précédents et se termine par un saut inconditionnel : JMP, $(XY)_2$.

Programme .

Adresse	Code mnémonique	Code hexadécimal	
0F00	NOP	08	
0F01	LDI	C4	
0F02	00	00	F0 = 0
0F03	CAS	07	
0F04	CSA	06	
0F05	ANI	D4	scrutation
0F06	10	10	et test
0F07	JZ	98	de SA
0F08	$(XY)_1$	F8	
0F09	LDI	C4	
0F0A	01	01	F0 = 1
0F0B	CAS	07	
0F0C	JMP	90	saut
0F0D	$(XY)_2$	F6	inconditionnel

Calcul des sauts.

$(XY)_1$: $(0F08)_H + (XY)_1 = (0F00)_H$

soit $(XY)_1 = (-8)_D = (F8)_H$

$(XY)_2$: $(0T0D)_H + (XY)_2 = (0F03)_H$

soit $(XY)_2 = (-10)_D = (F6)_H$

Exploitation.

— Lancer le programme.

— Actionner SA, la lampe doit s'allumer ou s'éteindre suivant l'état de SA.

Autres possibilités.

— Utiliser SB comme entrée, F1 ou F2 comme sortie.

— *Fonction ET.*

Utiliser les deux entrées SA et SB et une sortie (par exemple F0) pour réaliser une fonction ET entre SA et SB : lampe allumée seulement si SA et SB sont à 1, ce qui correspond par exemple à l'organigramme de la fig. 14.

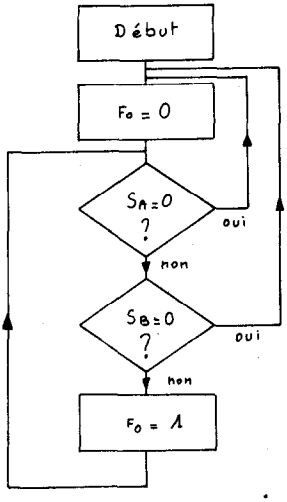


Fig. 14

On procède de même pour d'autres fonctions logiques.

— *Fonction monostable :* une seule entrée, une seule sortie, mais temporisée :

- si SA = 0 lampe éteinte,
- si SA = 1 la lampe s'allume pendant T secondes (SA = 0 est alors sans effet),
- organigramme : voir fig. 15.

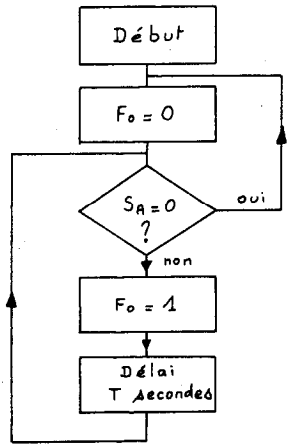


Fig. 15

On reconnaît l'organigramme de la fig. 13, mais avec un délai de T secondes.

Le μP SCMP possède une instruction de délai sur deux octets : DLY, (XY), le deuxième octet indiquant la durée Δt de la temporisation selon la relation :

$$\Delta t = [13 + 2(AC) + (2 + 2^9)(XY)_D] \times 1\,\mu\text{cycle}$$
$$\simeq [514 \times (XY)_D] \times 1\,\mu\text{cycle}.$$

Si l'horloge a une fréquence de 4 MHZ, la durée d'un micro-cycle est de 1 μs,

donc $\Delta t \simeq [514 \times (XY)_D] \times 1\,\mu s$

si (XY) = valeur maximum = $(FF)_H = (255)_D$

alors Δt max. $\simeq$ 0,131 s.

Pour obtenir des temporisations plus longues, on produit N délais Δt tels que $T = N\,\Delta t$.

Par exemple, pour T = 10 s avec Δt = 0,1 s,

on a besoin de N = 100

pour Δt = 0,1 s (XY) = $(194)_D$ soit $(XY) = (C2)_H$.

La durée T est obtenue par N tours de boucle de durée élémentaire Δt. On utilise une mémoire M servant de compteur. Au départ, M contient N. A chaque tour, on décrémente M et on teste si M = $(00)_H$ suivant l'organigramme de la fig. 16.

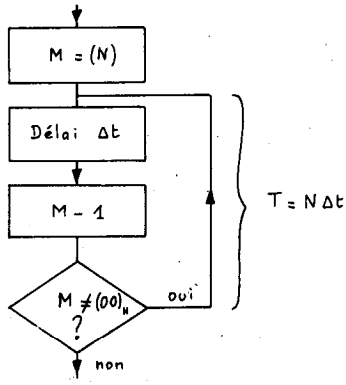


Fig. 16

Remarques.

— On peut utiliser le registre extension (EX) comme mémoire M.

— Pour retrancher 1 au contenu de M, on lui additionne $(-1)_D$ c'est-à-dire $(FF)_H$: il y a apparition d'une retenue (CY/L) que l'on doit faire disparaître (pour ne pas perturber le fonctionnement au prochain tour de boucle) par l'instruction CCL (mise à zéro de la retenue).

— On peut aussi utiliser une mémoire quelconque que l'on adresse au moyen d'un pointeur. Alors on simplifie l'opération de soustraction en utilisant une instruction de décrémentation du contenu de la mémoire : DLD/Pi, (déplacement). De plus, le contenu de M après décrémentation est placé dans l'accumulateur.

Les applications de la fonction temporisation sont nombreuses : mise en marche de moteurs pour des durées déterminées, affichages clignotants (feux de croisement), générateurs de signaux non conventionnels et de durées programmables, boîtes à musique (chaque note est un signal rectangulaire dont on règle la période par un délai Δt), etc.

Signalons, pour terminer, que l'on peut réaliser des automatismes simples avec des dispositifs attrayants : trains électriques miniatures, etc.

ANNEXES

Annexe n° 1.

La numération binaire ou de base 2 emploie deux caractères 0 et 1, appelés aussi bits ou digits.

Par exemple, le nombre binaire 1 011 représente en décimal : $(1\,011)_B = (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_D = (11)_D$.

Un octet est un ensemble de 8 bits, format désormais classique de l'information circulant dans la majorité des systèmes à microprocesseur. Le bit le plus à gauche est celui qui a le « poids » le plus fort (2^7), celui le plus à droite est celui qui a le « poids » le plus faible (2^0).

Remarque.

On pourra écrire un octet en binaire, en décimal ou en hexadécimal en précisant B pour binaire, D pour décimal et H pour hexadécimal. Exemple :

$$(1100\,0111)_B = (C7)_H = (199)_D.$$

Annexe n° 2.

La numération hexadécimale ou de base 16 emploie 16 caractères notés 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Par exemple, le nombre hexadécimal 8CA représente en décimal :

$$(8CA)_H = (8 \times 16^2 + 12 \times 16^1 + 10 \times 16)_D = (2\,250)_D.$$

En micro-informatique, la numération hexadécimale n'est pas employée en tant que telle, mais comme simplification de l'écriture binaire. Remarquons en effet qu'un ensemble de 4 bits offre $2^4 = 16$ possibilités. Il est donc possible de coder les 16 caractères hexadécimaux sur ces 4 bits, ce qui est représenté sur le tableau suivant.

Décimal	Binaire	Hexadécimal
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
10	1 0 1 0	A
11	1 0 1 1	B
12	1 1 0 0	C
13	1 1 0 1	D
14	1 1 1 0	E
15	1 1 1 1	F

Pour simplifier l'écriture d'un nombre binaire, on regroupe les bits par 4, à partir de la droite. Le cas de l'octet est le plus fréquent : 8 bits sont « condensés » en deux caractères hexadécimaux.

Exemple :

$$\underbrace{(1010}_{A}\ \underbrace{0101)_B}_{5} = (A5)_H$$

Annexe n° 3.

Fonctions logiques de base (voir fig. 17).

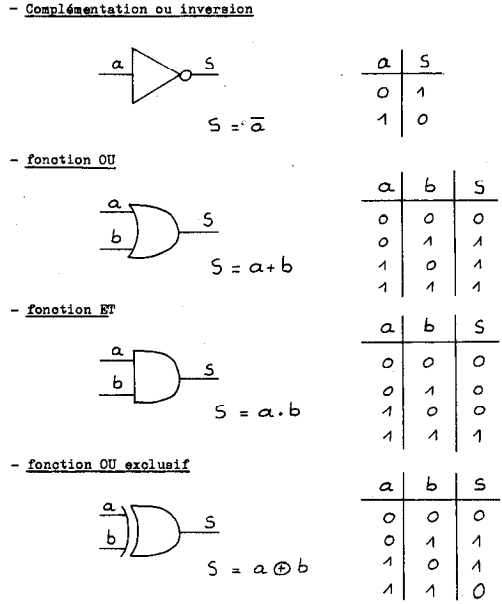


Fig. 17

Annexe n° 4.

Le système BCD : binaire codé décimal.

Le système décimal utilise 10 caractères qui peuvent être codés sur 4 bits comme les caractères hexadécimaux (voir tableau précédent). La différence réside dans le fait que six possibilités de combinaison ne sont pas utilisées et ignorées par l'unité arithmétique (celles correspondant aux caractères A, B, C, D, E, F donc aux valeurs décimales 10, 11, 12, 13, 14 et 15). Ce système facilite les opérations arithmétiques en évitant la fastidieuse conversion binaire décimale.

Exemple :

Soit à additionner les deux octets $(23)_H$ et $(47)_H$:

- si on fait une addition binaire classique, le résultat est $(6A)_H$,
- si on fait une addition dans le système BCD, on obtient (70), c'est-à-dire la valeur décimale.

Annexe n° 5.

Représentation de nombres algébriques sur un octet.

Soit A un nombre binaire écrit avec 7 bits, on peut ainsi représenter $2^7 = 128$ nombres, de $(0)_D$ à $(127)_D$.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
a7	a6	a5	a4	a3	a2	a1	a0

Soit A = a6 a5 a4 a3 a2 a1 a0 avec a7 = 0.

Considérons $\overline{A} = \bar{a}6\ \bar{a}5\ \bar{a}4\ \bar{a}3\ \bar{a}2\ \bar{a}1\ \bar{a}0$ ($\overline{A}$ est le complément à 1 de A).

Remarquons que :

$$ai + \bar{a}i = 1 \quad \text{donc} \quad A + \overline{A} = 111\,1111$$

et :

$$1 + A + \overline{A} + 1\,000\,0000 = (2^7)_D \quad \text{ou} \quad 2^7 - A = \overline{A} + 1.$$

$2^7 - A$ s'appelle le complément à 2^7 de A, il s'écrit sur 8 bits (on dit aussi complément à 2 de A).

Convenons que $2^7 - A$ représente $(-A)$ et que le bit a7 représente le signe :

a7 = 0 pour le signe +, a7 = 1 pour le signe —.

Calcul de (— A).

On applique la relation $(-A) = \overline{A} + 1$, c'est-à-dire que l'on inverse tous les bits de A, puis on leur ajoute 1.

Exemple : représentation de $(+1)_D$ et $(-1)_D$:

$(+1)_D$ s'écrit A = 0000 0001 = $(01)_H$

$\overline{A}$ = 1111 1110

$(-1)_D$ s'écrit $\overline{A} + 1$ = 1111 1111 = $(FF)_H$.

Remarques.

— Avec ce procédé, sur un octet, on peut représenter 128 nombres positifs, de 0 à $(+127)_D$ et 128 nombres négatifs, de $(-1)_D$ à $(-128)_D$.

— Sans convention de signe, sur un octet, on peut représenter : $2^8 = 256$ nombres, de $(0)_D$ à $(255)_D$.

Annexe n° 6.

DÉFINITIONS.

Microprocesseur.

Le processeur est l'unité centrale de traitement d'un ordinateur, c'est son « cerveau », son rôle est de commander l'action des autres organes de l'ordinateur et d'exécuter le programme situé en mémoire. Un processeur réalisé sous la forme d'un circuit intégré s'appelle un microprocesseur.

Ordinateur.

Un ordinateur simple se compose d'une unité centrale, d'une mémoire et de dispositifs d'entrées/sorties (clavier, écran vidéo, etc.) (voir fig. 18).

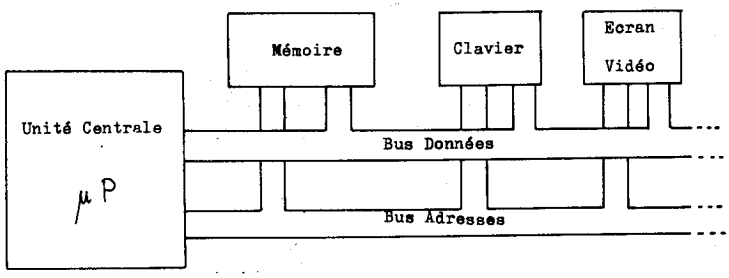


Fig. 18

Les données (valeurs numériques binaires, codes d'instructions) transitent par un ensemble de fils appelé bus, chaque fil véhiculant un bit de donnée. Les données sont aiguillées vers le périphérique correspondant (mémoire, clavier, etc.) au moyen d'un ensemble de fils appelé bus d'adresses. La combinaison des bits véhiculés par ces fils compose une adresse ou numéro (avec 16 fils, on a 2^{16} combinaisons, soit 65 536 adresses différentes).

Le microprocesseur peut être utilisé dans un micro-ordinateur, mais il a été conçu dans un but plus modeste, pour la réalisation d'un automate programmé : système de régulation, machine à laver automatique, etc.

Le programme et la mémoire.

Le programme est stocké en mémoire. On distingue des mémoires mortes (ROM ou PROM) où le programme a été défi-

nitivement écrit et des mémoires vives (RAM) où le programme peut être écrit et effacé à volonté. Une case mémoire à un format de 8 bits, elle contient donc un octet, elle est repérée par son numéro ou adresse.

C'est le programme qui indique au microprocesseur la succession des opérations logiques qu'il a à effectuer. Le programme est traité séquentiellement, instruction après instruction, le rythme est imposé par le compteur ordinal (PC) lui-même cadencé par une horloge interne.

Automate programmé.

Un automate fonctionnant en logique programmée est composé d'un microprocesseur, d'un ou plusieurs boîtiers mémoires, de quelques circuits d'entrée/sortie, mais généralement sans clavier ou affichage. Le programme de fonctionnement a été écrit définitivement dans une mémoire morte, les interventions sont quasiment impossibles : il s'agit d'un produit figé.

Au cours de l'étude et de la réalisation du prototype, il faut pouvoir changer des éléments, tester des programmes, les modifier à volonté. Pour cela, il faut disposer de l'équivalent d'un simulateur logique : un simulateur de logique programmée appelé aussi kit d'initiation ou système de développement. Ces dispositifs permettent d'écrire un programme grâce aux touches hexadécimales et de fonctions, de le vérifier grâce à un affichage, de gérer avec ce programme des entrées et des sorties logiques, c'est-à-dire de faire fonctionner un dispositif réel, tout en ayant ensuite la possibilité de s'adapter à une nouvelle application.

Annexe n° 7.

Le microprocesseur SCMP II.

Ce microprocesseur est produit par la Société National Semiconductor. Il est du type technologie MOS canal N avec une seule tension d'alimentation (+ 5 V), c'est un μP à 8 bits de données. Son intérêt réside dans son faible coût, l'intégration de certaines fonctions comme par exemple les entrées et les sorties logiques, la présence d'instructions spéciales (par exemple le délai).

Pour la programmation, on retiendra la présence d'un accumulateur (AC) de 8 bits, d'un compteur ordinal (PC) de 16 bits, de trois registres pointeurs (P1, P2 et P3) de 16 bits, d'un registre d'extension (EX) de 8 bits avec une entrée et une sortie de type série (Sin et Sout) et d'un registre d'état (SR) avec deux entrées logiques (SA et SB ; SA permettant les interruptions) et trois sorties logiques (F0, F1 et F2). On consultera la notice constructeur pour plus de détails.

Annexe n° 8.

Les instructions du microprocesseur SCMP II.

On peut les classer en instructions immédiates sur deux octets, en instructions sur le registre extension (un octet), en instructions indexées ou auto-indexées par rapport au compteur ordinal ou aux pointeurs sur deux octets [le deuxième octet indique le déplacement : c'est un nombre binaire algébrique, compris entre $(-128)_D$ et $(+127)_D$], enfin des instructions diverses sur un ou deux octets.

De nombreuses instructions ont déjà été décrites, pour les autres se référer au manuel fourni avec chaque kit d'initiation.

Annexe n° 9.

Description du kit d'initiation EMR.

Il est référencé : « Unité centrale, n° UC 1 003 » et diffusé par la Société EMR (185, avenue de Choisy, 75013 Paris).

Il se présente sous la forme d'un circuit imprimé comprenant, outre la partie électronique, un clavier de 20 touches et 6 afficheurs, 7 segments (fig. 19).

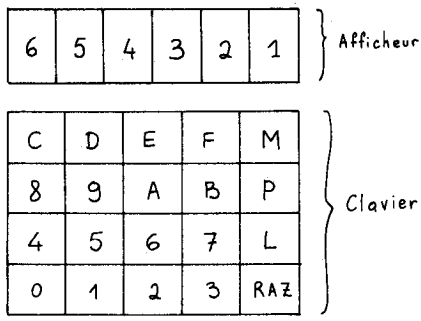


Fig. 19

Les afficheurs 1 et 2 sont réservés aux données, les suivants (3, 4, 5 et 6) aux adresses. Par programme, on peut les affecter à d'autres usages.

Le clavier comprend 16 touches hexadécimales (notées de 0 à F) et 4 touches de fonctions :

— RAZ : remise à zéro générale, provoque l'apparition de 6 tirets sur les afficheurs,

— P indique au μP que l'on veut composer une adresse (4 tirets à gauche, 2 points à droite), on écrit ensuite l'adresse désirée de 4 chiffres,

— M : quand une adresse est affichée :

1) indique le contenu de la case mémoire correspondante, ou sa modification en écrivant un nouvel octet,

2) permet, par un nouvel appui, d'incrémenter d'une unité l'adresse précédente,

— L : quand une adresse est affichée, permet l'exécution du programme, à partir de cette adresse.

La mémoire vive (RAM) où l'on écrit le programme utilisateur est composée de 512 octets, entre les adresses $(0E00)_H$ et $(0FFF)_H$.

Pour les manipulations avec les entrées et les sorties logiques, on doit respecter un minimum de précautions :

— bien que protégées, les entrées peuvent être détruites par des charges électriques statiques : prévoir si nécessaire un étage logique « tampon », non inverseur,

— les sorties peuvent débiter 100 µA à l'état 1 (5 volts) et 2 mA à l'état 0 (0 volt) : prévoir un amplificateur de courant, par exemple à transistor, quand cela sera nécessaire.

Remarque.

On trouve sur le marché deux autres kits d'initiation avec le SCMP II :

— le modèle MK 14 diffusé par de nombreux revendeurs,

— un système complet décrit et diffusé par la revue « Elektor ».

BIBLIOGRAPHIE

Les ouvrages relatifs aux microprocesseurs sont, à présent, très nombreux. On retiendra un ouvrage de caractère général, consacré aux ordinateurs :

— « *Structure et fonctionnement des ordinateurs* », de J.-P. MEINADIER. chez Larousse.

Pour le microprocesseur SCMP, on retiendra :

— « *La micro-informatique* ». Cours polycopié en trois tomes de G. LELARGE et J.-L. PLAGNOL, diffusé par la Société EMR (déjà citée).

— Notice sur le µP SCMP II (INS 8 060), diffusée par National Semiconductor.