9/98

République Algérienne Démocratique et Populaire
Ministère de l'enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

D.E.R DE GENIE ELECTRIQUE ET INFORMATIQUE

FILIERE : ELECTRONIQUE

PROJET DE FIN D'ETUDES

Pour l'obtention du diplôme d'ingénieur d'état
en Electronique

Thème

CONCEPTION MICROPROGRAMMÉE D'AUTOMATISMES APPLICATION A LA CONCEPTION D'UN CONTROLEUR D'UNE PORTE DE SÉCURITÉ

Proposé et Dirigé par :
Mr A.FARAH

Présenté par :
Sofiane HAMADOUCHE
Abdenour BOUNSIAR

Promotion : *Septembre 1998*

E.N.P. 10, AVENUE HASSEN BADI - EL-HARRACH - ALGER

Dédicaces

Je dédie ce mémoire à mes chères parents

à mes frères et sœurs

à tout mes amis et proches.

H. Sofiane.

Je dédie ce mémoire à mes chères père et mère

à tout mes frères et sœurs

à mes amis et à tout mes proches .

B. Abdenour.

SOMMAIRE

CHAPITRE II : Machines séquentielles microprogrammée

CHAPITRE III : conception d'un automate de commande d'une porte de sécurité à code.

INTRODUCTION GENERALE

INTRODUCTION GENERALE :

L'automatisation d'un procédé ou d'une machine consiste à en assurer la conduite par un dispositif technologique qui est l'automate. Cette automatisation répond à plusieurs critères :

- Un critère sécuritaire : c'est la protection de l'accès à l'information.
- Un critère économique : c'est l'augmentation de productivité, l'amélioration de la qualité de produit, la diminution des coûts de production.
- A ces deux critères, s'ajoute celui de l'amélioration des conditions de travail, la sécurité était accrue et les tâches pénibles et répétitives supprimées.

Au début des années 50, les ingénieurs étaient déjà confrontés à des problèmes d'automates, leur composant de base était le relais électromagnétique à un ou plusieurs contacts. Les automatismes qu'ils concevaient comportaient plusieurs centaines, voir plusieurs milliers de relais.

Ce n'est que vers 1970 que l'apparition des C.I a amorcé une révolution dans la façon de concevoir les automatismes, ceux-ci apportant sous un faible volume, avec un faible consommation d'énergie, la possibilité de réaliser des fonctions plus complexes, avec une plus grande rapidité et avec un prix toujours décroissant.

Les méthodes traditionnelles de conception des systèmes logiques utilisant des connaissances théoriques réalisant le circuit par connexion et assemblage de portes ne sont pas très appropriées pour les technologies modernes. Le majeur différence est la complexité des systèmes que nous sommes maintenant capables de construire en un minimum d'espace et du coût .

Un problème important de conception est le type de technologie choisie pour réaliser le système logique et le but de la conception. Le choix final doit être décidé par le système à développer, en tenant en compte les facteurs de performances, volume de production, le matériel disponible, ...etc.

Le processus de conception doit dérouler en différents niveaux :

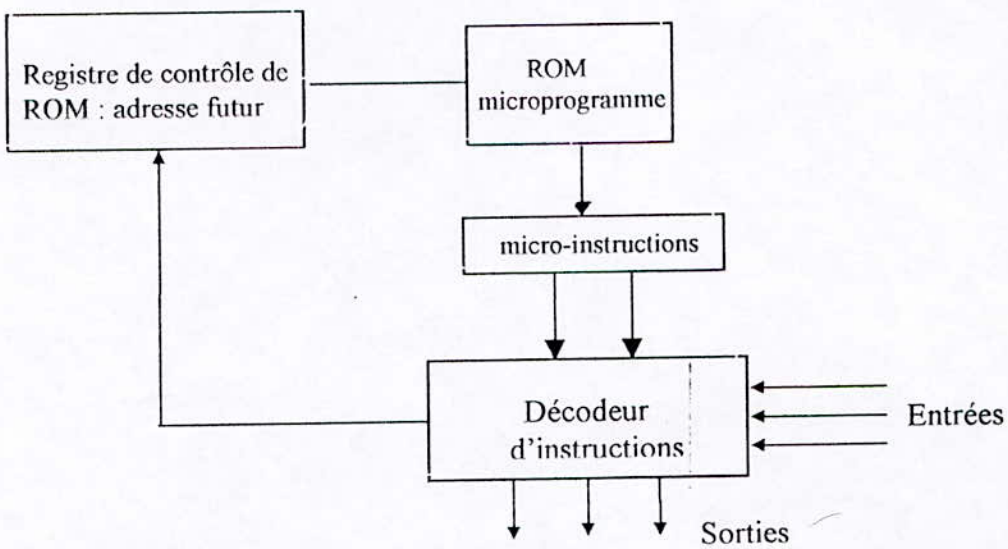
* description du comportement du système en terme de processus algorithmique.
* translation logique des différentes tâches et séquences pour les représenter sous forme de fonctions logiques.

* translation physique pour la représentation structurelle en circuits physique : implémentation.

Le concept de la microprogrammation à été proposé pour la première fois par WILKS pour la conception d'une unité de contrôle dans un ordinateur digital. Les signaux de contrôle de la machine interne sont appelés micro-instructions et l'ensemble de ces instructions qui sont logiquement des signaux qui ouvrent et ferment des portes, mettant des données dans des registres, signaux de lecture et d'écriture de mémoire, adresses de sélection des MUX,...etc.

Les mots de la ROM d'un microprogramme sont lus séquentièllement et les bits de sortie sont transmis, après décodage par un interpréteur aux périphériques, (partie opérative d'un automate, structure de données d'un ordinateur,...etc.)

L 'unité de microprogramme à base d'une ROM est représentée dans la figure suivante :



PRESENTATION DU TRAVAIL :

Cette mémoire est organisée en trois chapitres :

Au premier chapitre, nous présentons les méthodes de description des machines séquentielles ; nous commençons d'abord par la description par équation logique, la table des états ou des transitions, le graphe des états qu'il n'est qu'une forme plus compréhensible de la table des états, le grafcet et réseau de pétrie, diagramme de décision binaire et, enfin, la description par microprogramme mnémonique et binaire.

Le deuxième chapitre présente les machines séquentielles microprogrammée, les notions du sous programme, procédure, procédure avec paramètre seront détaillées à l'aide des exemples simples ; ce chapitre se termine par la mise en évidence du microprogramme incrémenté qui permet la restriction de la taille de la mémoire en largeur puisque il n'exige que la spécification d'une seule adresse.

Le chapitre trois est consacré à l'application de la méthode microprogrammée à la conception d'un automate de commande d'une porte de sécurité à code ; nous envisageons les deux cas :

- Cas où la comparaison se fait par hard à l'aide d'un comparateur.
- Cas où la comparaisons se fait par soft (microprogramme).

Nous terminons ce chapitre par la simulation sur ordinateur.

CHAPITRE I :

méthodes de description des machines séquentielles.

CHAPITRE I : METHODES DE DESCRIPTION DES MACHINES SEQUENTIELLES :

I- Introduction :

La conception d'automates logique destinés à prendre des décisions binaires pose à l'industriel de nombreux problèmes. Le cahier des charges est difficile à établir, les méthodes systématiques de synthèse et les représentations fonctionnelles de description ne sont applicables que pour un nombre de variables toujours insuffisant. Les solutions obtenues créent des fonctions internes auxquelles le concepteur peut difficilement attacher un sens physique.

Le réalisateur travaille alors en aveugle et l'utilisateur est dans l'ignorance totale du fonctionnement de l'appareil. La maintenance est donc difficile, et de ce fait coûteuse. Les modifications de spécification remettent souvent en question une trop grande part de l'ensemble ce qui engage alors l'industriel soit à y renoncer, soit à accoler des circuits supplémentaires sans revoir la structure globale du fonctionnement. On arrive ainsi à des redondances de matériel inutiles coûteuses et à des réactions imprévues d'un système que l'on ne maîtrise que partiellement.

Beaucoup sont les méthodes qui ont été développées pour décrire le comportement et l'évolution d'un circuit logique. Un système combinatoire peut être décrit par son équation logique, sa table de vérité, tableau de KARNAUGH, ou par le graphe de décision binaire. Le choix entre ces différentes méthodes se fait en fonction, surtout, de la simplification qu'on peut effectuer par une de ces méthodes. Le tableau de KARNAUGH est souvent utilisé pour un nombre de variables d'entrée inférieur à six.

En logique séquentielle, l'état de sortie du système n'est plus fonction des variables d'entrée seules, mais aussi des états présents du système. La description de ces systèmes séquentiels se fait par plusieurs méthodes : description mathématique par équations logiques, description tabulaire par la table de transition des états, description graphique par le graphe des états, de décision binaire ....etc, et description par microprogramme.

La méthode microprogrammée est très utilisée pour la description et la conception des petits automates à cause de sa simplicité et sa souplesse.

II. SYSTEMES COMBINATOIRES :

La théorie des systèmes logiques traite des assemblages que l'on peut former avec deux types d'éléments idéaux : les éléments combinatoires et les éléments séquentiels.

II.1.Définition : Un système combinatoire est caractérisé par le fait qu'une combinaison de variable d'entrée correspond à une et une seule combinaison de sortie, c'est-à-dire que l'état de sortie à un instant t quelconque est entièrement déterminé par l'état d'entrée à ce même instant. (voir figure (I.1)).

xi: variables d'entrée

zj : variables de sortie

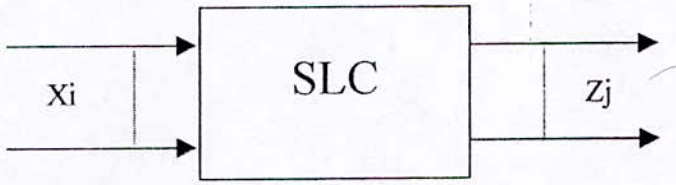


FIG.I.1 : système logique combinatoire

L'inverseur, les portes ET, OU , NAND ; les multiplexeurs et les démultiplèxeurs illustrent cette catégorie, aussi, l'assemblage de ces éléments (sans boucle de rétroaction) est un système logique combinatoire.

Dans la réalité, la réponse d'un système combinatoire, ou autre, n'est jamais instantanée. En effet, l'état de sortie n'apparaît qu'après un intervalle de temps de propagation tp. Chaque élément combinatoire est caractérisé par un temps de propagation appelé aussi : temps de réponse ou délai, qui dépends de la technologie utilisé (TTL, MOS, CMOS, ...). (voir figure (I.2)).

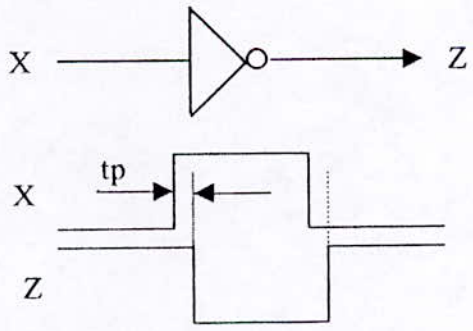


FIG. I.2: temps de propagation dans un inverseur

II.2. Description des systèmes combinatoires :

II.2.1. Equations logiques :

L'équation logique d'un système combinatoire c'est l'expression algébrique de la fonction de sortie de ce système en fonction des variables d'entrée; elle est exprimée en général sous forme de somme de produits.

Exemple 1:

$$Z = f(a, b, c, d) = a.b.c.d + \bar{a}.b.c.d + a.b.\bar{c}.d + \bar{a}.b.\bar{c}.d + a.\bar{b}.c.\bar{d} .$$

Les manipulations sur les expressions logiques se font en respectant les identités de l'algèbre de boule et la simplification de ces équations s'effectue à l'aide des équations suivantes :

$$a + \bar{a} = 1; \qquad a + a = a ; \qquad a . \bar{a} = 0 .$$

et les relations de De Morgan:

$$\overline{a + b} = \bar{a} . \bar{b} ; \qquad \overline{a . b} = \bar{a} + \bar{b} .$$

ainsi, la fonction précédente peut être simplifiée comme suit :

$$Z = b.c.d (a + \bar{a}) + b.\bar{c}.d (a + \bar{a}) + a.\bar{b}.c.\bar{d} .$$

$$Z = b.c.d + b.\bar{c}.d + a.\bar{b}.c.\bar{d} = b.d (c + \bar{c}) + a.\bar{b}.c.\bar{d} = b.d + a.\bar{b}.c.\bar{d}$$

Quand le nombre de variables augmente, cette méthode est difficile à utiliser.

II.2.2. Table de vérité :

La table de vérité donne la valeur d'une fonction binaire pour chacune des combinaisons des variables d'entrée. Il existe 2^n combinaisons de n variables binaires.

La table de vérité de la fonction de l'exemple précèdent s'exprime comme illustré dans la figure (I.3).

a	b	c	d	z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

FIG.I.3: Table de vérité de la fonction Z.

II.2.3. Tableau de KARNAUGH :

Le tableau de KARNAUGH à la différence de la table de vérité met l'information dans des carrés organisés de telle sorte que les carrés adjacents horizontalement n'aient qu'une variable qui soit différente . Il on est de même pour les carrés adjacents verticalement .Dés qu'un tableau de KARNAUGH est garni de uns et de zéros, il s'agit , pour obtenir l'expression algébrique sous forme de somme de produits, d'additionner logiquement les carrés qui renferme un 1.

Le tableau de KARNAUGH correspondant à notre exemple est représenté dans la figure :

La simplification par la méthode de KARNAUGH est très utilisée pour un nombre de variables inférieur à six; elle consiste à réunir tout les doublets ou quartets des uns des carrés adjacents . L'expression algébrique correspondante ne tient en compte que des variables inchangées. Ainsi pour la figure :I.4; les variables inchangées dans le bloc considéré sont b et d. Le un restant on le tient séparément des autres. A la fin on aura l'expression simplifiée suivante:

$Z = b.d + a.\overline{b}.c.\overline{d}$

cd \ ab	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	1	1	0
1 1	0	1	1	0
1 0	0	0	0	1

FIG.I.4: Tableau de KARNAUGH de la fonction Z.

II.2.4. Graphe de décision binaire :

Un BDD (Binary Decision Diagram) peut être représenté par l'assemblage de nœuds de la figure (I.5-a).

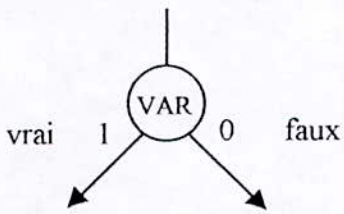


FIG.I.5-a: Nœud élémentaire d'un BDD.

La forme la plus utilisée des BDD est l'OBDD (Ordred BDD); c'est un BDD ordonné. l'OBDD de la fonction Z est représentée dans la figure ci-dessous.

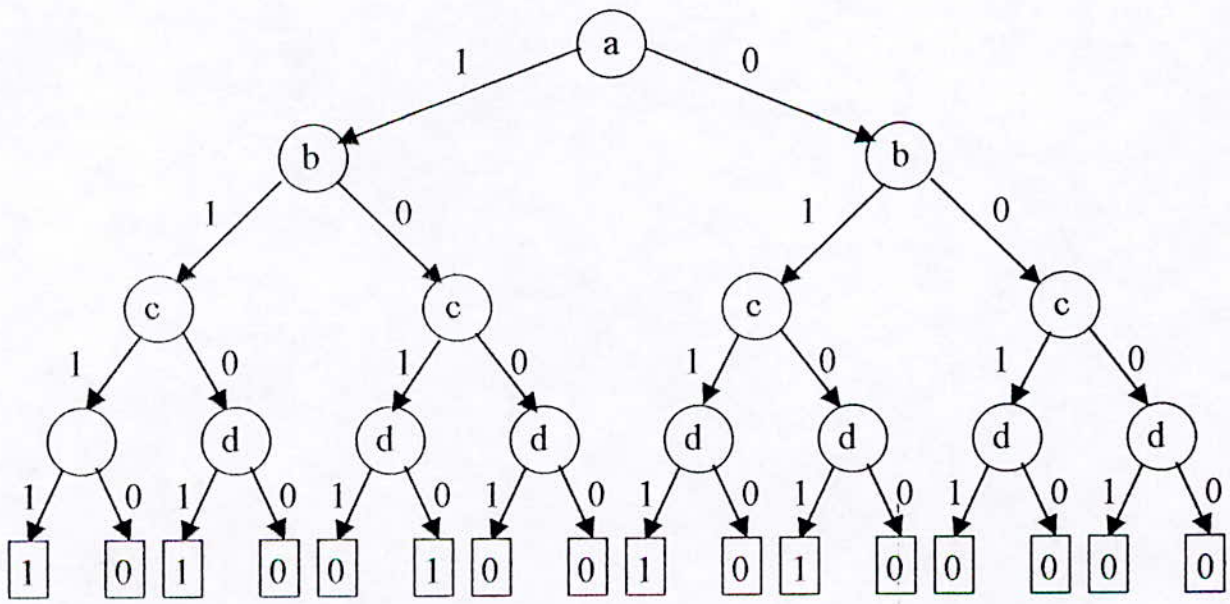


FIG.I.5-b :L'OBDD de la fonction Z.

III. Logique séquentielle :

III.1 : Définition :

Un automatisme logique est dit séquentiel si à une même situation des entrées correspondent diverses situations des sorties suivant l'état précèdent du processus et selon l'ordre dans lequel les états d'entrée ont été appliquées au système. La connaissance des entrées ne suffit plus à définir les sorties, il faut aussi connaître la situation intérieure de l'automate, pour cela le système doit mémoriser certaines « étapes » par lesquelles il est précédemment passé. On connaît qu'il ne soit pas nécessaire d'associer une telle « étape » à mémoriser à chacun des états d'entrée par lequel le système est passé.

Seuls certains de ces états d'entrée seront déterminants pour l'évolution du système et promettront de passer d'une étape à une autre.

Un système séquentiel est constitué d'un système combinatoire avec l'addition des propriétés de mémorisation et boucle de retour.

III.2. Machines séquentielles de MEALY et de MOORE :

Une machine séquentielle de Mealy est définie par le quintuplet MS (I,Q,Z,S,W) où

* I est un ensemble fini ($i_1,...i_n$) de combinaisons de E (ou vecteur d'entrée): alphabet d'entrée.

* Q est l'ensemble des états de MS.

* Z est un ensemble fini ($z_1,....,z_n$) de combinaisons de S (ou vecteurs de sortie) alphabet de sortie.

* δ est une application de I×Q dans Q appelée fonction état suivant.

* W est une application surjective de I×Q dans Z appelée fonction de sortie.

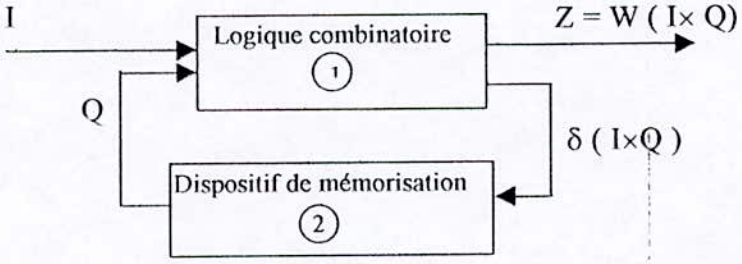


FIG.I.6 machine de Mealy.

Une machine séquentielle pour laquelle l'ensemble Q contient un nombre fini d'éléments est appelée machine à états finis.

le bloc(1) est un circuit combinatoire qui a partir d'un vecteur d'entrée appliqué I et de l'état présent Q élabore d'une part un vecteur de sortie Z ,d'où la fonction Z=W(I×Q). et d'autre état suivant Q par δ (I×Q).

Le bloc (2) a pour rôle de mémoriser état suivant jusqu'à ce qu'il devienne état présent à son tour. La machine séquentielle répondant à cette définition s'appelle machine de MEALY.

La réduction de la fonction de sortie W à l'application surjective W(Q)→Z donne lieu à un autre type de machine séquentielle : Machine de Moore.

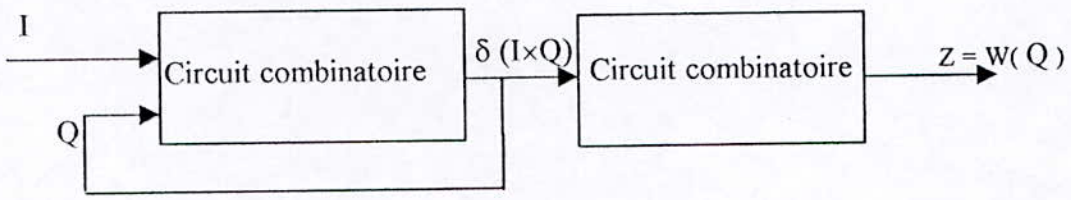


FIG.I.7 machine de Moore.

Il y a deux grandes classes des circuits séquentiels : synchrone (avec horloge) et asynchrone (sans horloge).

III.3. Systèmes synchrones et systèmes asynchrones :

III.3.1 : système synchrone :

Un système est dit synchrone lorsque son évolution est contrôlée. Elle peut l'être par les entrées elles-mêmes ou par un signal particulier appelé horloge . Cette dernière est la plus utilisée.

Dans le type de synchronisation par horloge, les entrées primaires sont des niveaux. L'exécution des entrées, des sorties et des états internes se fait à des intervalles de temps bien définis. contrôlés par la fréquence horloge du système qui, généralement, a une forme d'onde rectangulaire. Le système n'évolue que par l'action de l'horloge.

III.3..2. Systèmes asynchrones :

Un système asynchrone est soumis à des entrées de type à niveaux. L'évolution n'est contrôlée par aucune entrée particulière, de telle sorte que, pour un vecteur d'entrée appliquée, le système évolue jusqu'à ce que l'état présent et l'état suivant soient identiques. On dit alors que le système est dans un état stable. Le système évolue sous l'effet des entrées seulement.

IV - Description des systèmes séquentielles :

La méthode classique de représentation des circuits séquentiels est soit par l'utilisation des graphes des états ou bien la table des états (généralement appelée: la table des transitions des états) ou encore le modèle mathématique d'un automate fini qui est peu utilisé. La table des états est peut être la plus importante puisqu'elle est la méthode la plus commode pour la représentation des exigences du système avant la réalisation. En général, la table des états se déduit du graphe des états puisque la plupart des algorithmes de conception commencent par le graphe des états.

IV.1. Description par équations logiques :

La description des systèmes séquentiels par équations logiques consiste à déterminer les expressions algébriques des fonctions de sortie ($Z = W (I \times Q)$ pour le modèle de MEALY et $Z = W(Q)$ pour le modèle de MOORE) et d'états futurs $Y = \delta (I \times Q)$.

Exemple :

$$Z = W(I \times Q) = W [(X_0, X_1) \times (Y_0, Y_1)] = \overline{X}_1 . X_0 . Y_1 . Y_0 .$$

$$Y^+ = (Y_0^+ , Y_1^+) = \delta [(X_0, X_1) \times (Y_0, Y_1)]$$

$$= (\overline{X}_1 . \overline{X}_0 . \overline{Y}_1 . \overline{Y}_0 + \overline{X}_1 . \overline{X}_0 . Y_1 . Y_0 + X_1 . X_0 . Y_1 . \overline{Y}_0 , \overline{X}_1 . \overline{X}_0 . \overline{Y}_1 . Y_0 + \overline{X}_1 . \overline{X}_0 . Y_1 . \overline{Y}_0 + X_1 . X_0 . Y_1 . \overline{Y}_0)$$

Le système tel définit conçu pour détecter la l'occurrence de la séquence des paires d'entrée (00, 00, 11, 10).

IV.2- Description tabulaire :

Le développement de la table de transition pour la spécification du circuit original, formule logiquement le problème dans la même méthode comme la table de vérité pour un circuit combinatoire. elle est, en effet, une représentation mathématique abstraite d'un circuit séquentiel; elle trouve ces origines dans la table des fonctions utilisées dans la théorie des groupes pour décrire les opérateurs binaires.

Dans la table des états on représente les entrées, les sorties, les états présents et les états futures.

Exemple 2:

La table des états d'une machine qui détecte la l'occurrence de la séquence des paires d'entrée (00, 00, 11, 10).

On a quatre états internes et deux variables d'entrée et une seule sortie. On choisi le codage binaire direct pour coder toutes les variables.

variables		état présent		état futur		sortie
X 1	X0	Y1	Y0	Y1	Y0	Z
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	1
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	1	1	0
1	1	1	1	0	0	0

FIG.I.8 : table des états de l'exemple 2.

VI-3 : Description graphique :

VI.3.1 : graphe des états :

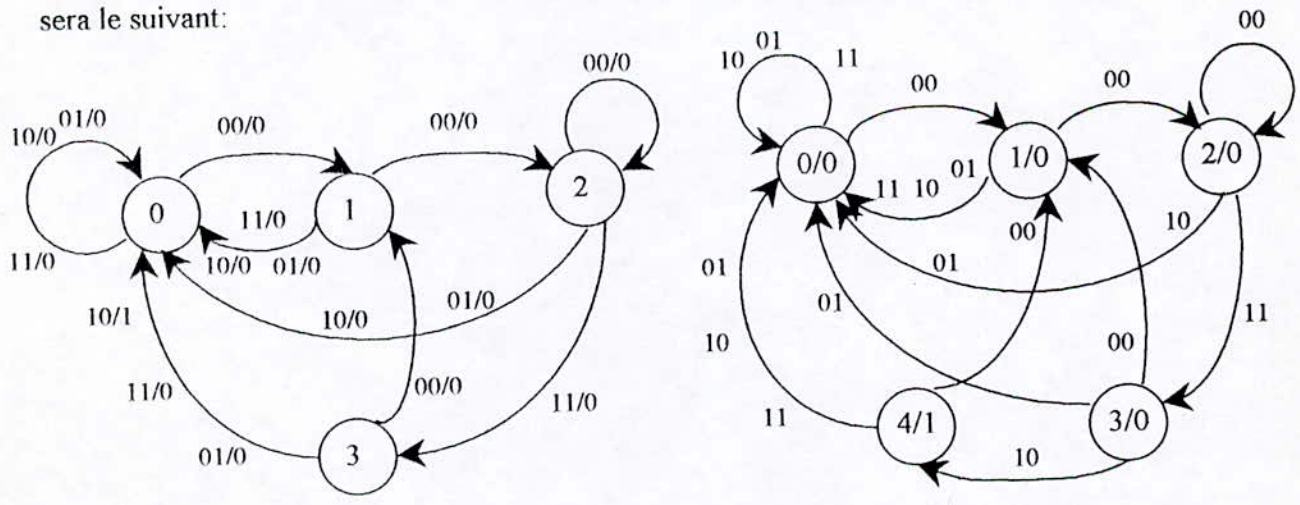
Pour des systèmes compliqués, la table des états est parfois difficile à construire. Il n'y a pas des techniques établies, le processus est surtout intuitive et relié à des expériences. Le graphe des états contient exactement les mêmes informations que la table des états mais dans une forme plus compréhensible.

Le graphe des états est un graphe orienté; les états de graphe sont représentes par des cercles(nœuds) avec des lignes fléchés entre eux montrant les chemins de transition. Il existe deux type de graphe des états, qui correspondent aux modèle de MEALY et de MOORE. Dans le premier cas, chaque chemin est indiqué avec l'entrée qui cause la transition et la sortie résultante; le nœud contient le symbole ou code de l'état interne. Le modèle de Moore diffère de celui de Mealy; chacun des chemins est indicé avec les entrées qui cause la transition, les nœud contiennent les codes des états et l'état de sortie, c'est-à-dire l'état de sortie est fonction des états internes. (voir figure I.9). Quand les états initiaux et finals sont les mêmes, on appelle le chemin de transition une *boucle* .

FIG. I.9 : a) modèle de MEALY b) modèle de MOORE

Les deux modèles peuvent être utilisés pour représenter les machines séquentielles, le choix entre les deux modèles ce fait par des préférences personnelles et de commodité. Une fois le graphe des états du circuit séquentiel est produit et testé, ça sera facile de le convertir en table des états.

Retrouvons l'exemple de la table des états précédentes, le graphe des états correspondant sera le suivant:



b) : model de MEALY. b) : model de MOORE.

FIG.I.10 : graphe des états de l'exemple 2.

La table des états présente, par rapport au graphe des états, l'avantage important de suggérer les évolutions non encore envisagées. Elle est donc principalement utile lors de l'établissement des spécifications. Par contre, le graphe illustre mieux les évolutions différentes d'un système déjà complètement défini, surtout si on est parvenu à le rendre planaire (pas de croisement entre les liaisons), par une disposition judicieuse des différentes étapes les unes par rapport aux autres.

IV.3.2. Réseaux de pétri :

IV.3.2.1. Définition :

Parmi les approches de graphes orientés qui ont trouvé une application considérable dans les descriptions et analyse des systèmes digitaux, on trouve les réseaux de pétri. Un réseau de pétri est une représentation graphique formelle et abstraite de la circulation d'informations dans un système, il se compose de deux types de nœuds, les places représentées par des cercles, et les transitions représentées par des tirets, connectés entre eux par des arcs orientés. Chaque arc connecte une place a une transition ou vice versa ; dans le premier cas les places sont appelées place d'entrée et Dans le second cas des places de sortie de la transition.

En plus de représenter les conditions statiques d'un système, le comportement dynamique peut être visualisé par des marques amovibles (appelés marques ou jetons) d'une place à une autre à travers le réseau. On représente la présence d'une marque par un point noire dans le cercle de la place. (voir figure (I.11)).

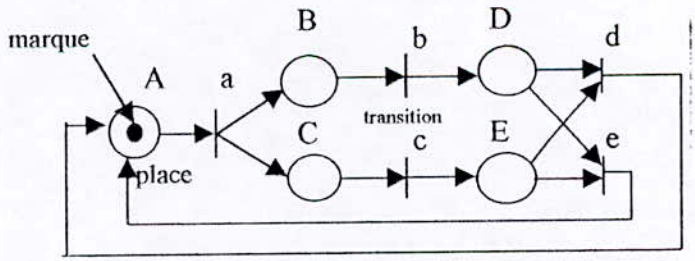


FIG.I.11

Le passage à travers le réseau d'un marquage à un autre correspondant à des changements d'états, est déterminé par le fonctionnement des transitions selon les règles suivantes :

- Une transition est activée si chacune de ces places d'entrée porte une marque.
- Toute transition active doit être franchie.

- Une transition est franchie par le transfert des jetons des places d'entrée vers les places de sortie ; ce franchissement veut dire que instantanément les entrées de la transition sont évidés et toutes ses sorties remplies.

Notons que les transitions ne peuvent pas être franchies simultanément, et que seulement une transition peut avoir lieu à un instant donné.

La figure(I.11) donne une représentation typique d'un réseau de pétri , où les places correspondent aux conditions du système qui doivent être satisfaites pour qu'une transition ai lieu.

Un réseau de pétri ordinaire R peut être représenté par un ensemble de places P, un ensemble de transitions T , une relation d'incidence avant I_α définie dans P × T , une relation d'incidence arrière I_β définie dans T×P et un marquage initial M_0 qui précise le nombre de marqueurs ou de jetons contenus à l'instant initial dans chacun des éléments de P. Un réseau de pétri ordinaire se définit donc par un cinq-tuplet :

$$R=(P, T, I_\alpha , I_\beta , M_0).$$

Pour décrire une machine séquentielle, un réseau de pétri est interprété par l'association aux places et aux transitions de prédicats logiques, fonctions des événements d'entrées et de sorties de la machine à représenter.

Pour un réseau ordinaire une transition ti est validée si toutes les places pj telle que $(p_j, t_i) \in I_\alpha$ possèdent au moins un marqueur .Elle est franchie si l'événement associé à ti se produit ; un marqueur est alors enlevé à chacune des places $p_{j,}$, tandis qu'un marqueur est ajouté à chacune des places pk tel que $(t_i, p_k) \in I_\beta$.

IV.3.2.2. Notion de conflit :

Lorsqu'il existe une place d'entrée commune à deux ou plusieurs transitions validées simultanément il n'est pas possible d'appliquer la règle de franchissement énoncée précédemment. Les transitions validées sont alors dites en conflit pour le marquage correspondant. L'apparition d'un conflit entraîne l'arrêt de l'évolution du marquage d'un réseau, il est nécessaire dans ce cas de rendre prioritaire une des transitions validées. Le franchissement de cette transition modifie le marquage et supprime la validation des autres transitions.

Exemple :

Les transitions t_1 et t_2 du réseau élémentaire donné par la figure (I.12-a) sont en conflit pour le marquage initial plaçant un marqueur en p_1, p_2 et p_3 ; p_2 étant une place d'entrée commune aux transitions t_1 et t_2 (on dit que la place p_2 est partagée). Si on suppose la transition t_2 prioritaire, celle-ci peut être franchie et mettre fin à la validation de l'autre (t_1). Le marquage résultant est représenté par la figure (I.12-b):

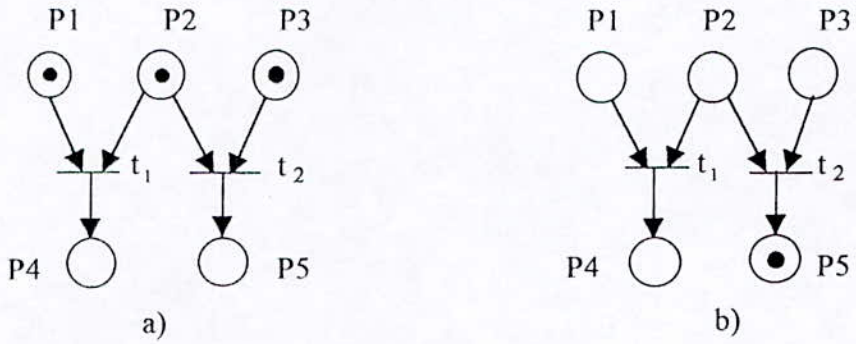


FIG.I.12

IV.3.2.3. Etude du marquage d'un réseau :

L'application séquentielle à un réseau de pétri de la règle de validation et de franchissement d'une ou d'un ensemble de transitions permet la détermination de l'ensemble des marquages qu'il est possible d'atteindre à partir d'un marquage initial M_0 . Nous dénoterons $M_0^1 = \{M_0, M_1, ...\}$ l'ensemble de ces marquages. Cet ensemble accessible à partir de M_0 dépend à la fois de la structure du réseau et du marquage initial M_0 .

Soit M_i un marquage sensibilisant la transition t_i et M_j le marquage résultant alors du franchissement de t_i .Nous écrivons alors :

$$Mi \xrightarrow{t_i} Mj$$

Soit σ une séquence finie de transitions t_i, t_{i+1}, t_{i+2},.....t_{i+k} appartenants à T ; on dit que σ est une séquence de franchissements franchissables à partir de M_i si et seulement si il existe des marquages M_{i+1}, M_{i+2},.....M_{i+k+1} tels que :

$$M_i \xrightarrow{t_i} M_{i+1}\,,\ M_{i+1} \xrightarrow{t_{i+1}} M_{i+2} \ldots\ldots\ldots\ldots\ldots\ldots M_{i+k} \xrightarrow{t_{i+K}} M_{i+k+1}$$

Nous écrivons alors :

$$Mi \xrightarrow{\sigma} M_{i+k+1}.$$

La classe des marquages conséquent M_0 est l'ensemble des marquages accessibles à partir de M_0 par une séquence de franchissements.

$$M_i \in M_0^1 \Leftrightarrow \exists \sigma , M_0 \xrightarrow{\sigma} M_i$$

IV.3.2.4. Propriétés définies sur les réseaux de PETRI (RdP) :

Si on peut classer les réseaux de PETRI suivant leurres structures, on peut également les distinguer en considérant certaines propriétés que leurs confère leur marquage initial M_0.

Soit R un réseau de pétri et M_0 sont marquage initial, M_0^1 sera la classe des marquage conséquents correspondante. Alors nous aurons les types de réseaux suivants:

1.Reseau sauf pour un marquage initial donné :

R est sauf pour M_0 si et seulement si tout marquage appartenant à M_0^1 est tel qu'il y a au plus un jeton (marquage) dans chaque place de R.

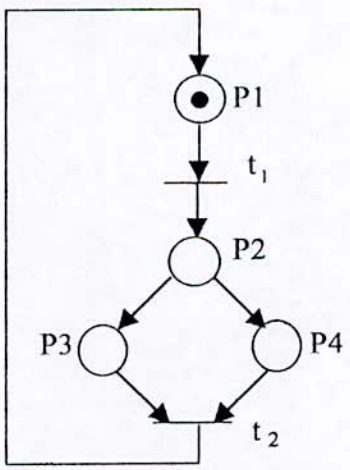


FIG.I.13 : réseau saut.

Autrement dit : un réseau est sauf ou sain pour M_0 si quel que soit le marquage obtenu à partir de M_0 par une séquence finie de franchissements, aucune place ne possède plus d'un marqueur .

Remarque : *L'ensemble M_0 ne peut comprendre qu'un nombre fini de marquages saufs puisque le nombre de places d'un réseau est fini.*

2. Réseau vivant pour un marquage initial donné :

Un réseau est dit vivant pour M_0 si toute transition du réseau peut être validée et franchie par une séquence finie de franchissements.

Autrement dit : si pour toute transition t de R et pour tout marquage Mi de M_0 il existe une séquence de franchissements qui franchit t à partir de Mi alors R est dit vivant pour M_0. Un exemple de réseau vivant est donné par la figure(I.14), celui de la figure(I.15) est non vivant.

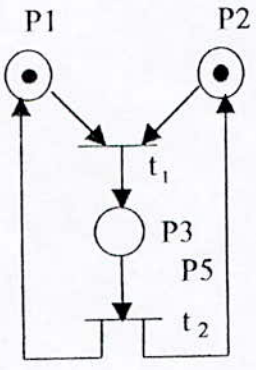


FIG.I.14 : réseau vivant.

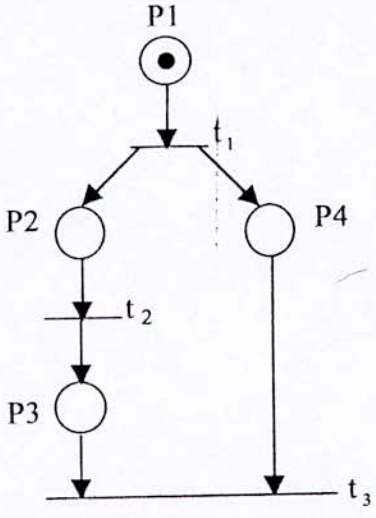


FIG.I.15 : réseau non vivant.

Le réseau de la figure(I.15) n'est pas vivant puisqu'on obtient le marquage vide après avoir exécuter une séquence complète de commandes. Pour le rendre vivant il suffit de le reboucler pour que la place P1 devienne place de sortie de la transition t_3 à la quelle aucune action n'est associée.

Remarque: *Le fait qu'un réseau marqué soit vivant a deux conséquences importantes en pratique : d'une part, cela entraine l'absence de blocage (marquage accessible à partir du quel aucune transition n'est franchissable), d'autre part, il est certain qu'aucune partie de la commande ne deviendra inaccessible après une certaine séquence de franchissements. Un réseau vivant et sauf est dit conforme (figure :I.16).*

3.Réseau pseudo-vivant pour un marquage initial :

Un réseau est pseudo-vivant pour le marquage M_0 , si à partir de tout marquage appartenant à M_0, il existe au moins une transition franchissable (voir figure(I.17)).

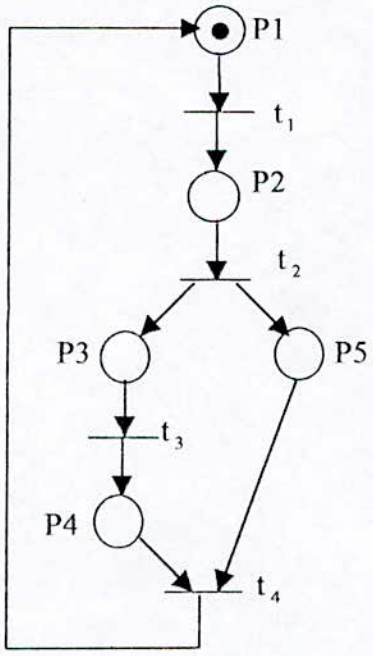


FIG.I.16 : réseau conforme.

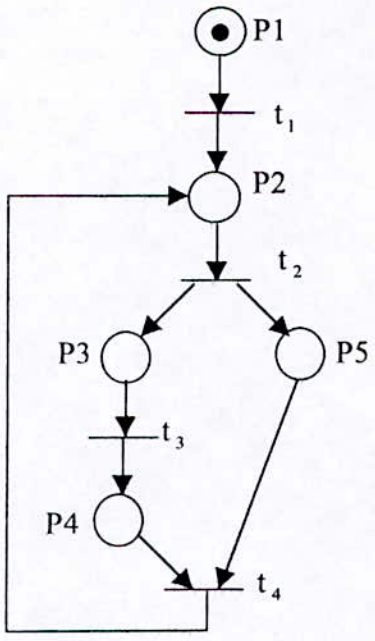


FIG.I.17 : réseau pseudo-vivant.

4.Réseau propre pour un marquage initial :

Un réseau de pétri est dit propre pour M_0 si et seulement si quel que soit $Mi \in M_0^1$ il existe une séquence de franchissements σ finie telle que $Mi \xrightarrow{\sigma} M_0$ (figure:I.18).

***Remarque** : un RDP conforme pour M_0 n'est pas nécessairement propre pour ce marquage (figure:I.19 .*

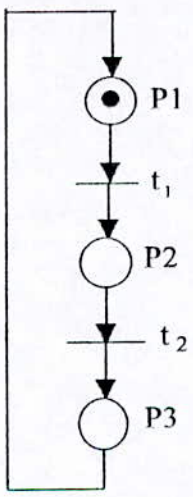


FIG.I.18 : réseau propre.

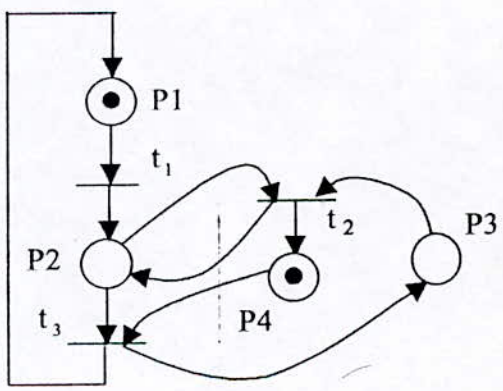


FIG.I.19 : réseau conforme mais pas propre.

En général les systèmes de commandes que l'on considère ont un fonctionnement cyclique, c'est à dire lorsqu'ils ont terminé la tache pour la quelle ils ont été conçus, ils sont prêts à la recommencer. M_0 représentant l'état initial de la commande , il est donc claire que le RDP représentant un système de commande doit être propre.

IV.3.3. GRAFCET:

IV.3.3.1. Définition :

Un « graphe fonctionnel de commande etapes-transitions » (GRAFCET) est un mode de représentation et d'analyse d'un automatisme , particulièrement bien adapté aux systèmes dont les évolutions peuvent s'exprimer séquentièllement, c'est à dire dont la décomposition en étapes est possible.

Ce mode de représentation présente un certain nombre d'avantages par rapport aux moyens décrits antérieurement (graphes et tables d'états).

- Il est indépendant de la matérialisation technologique de l'automatisme, que celle-ci soit câblée ou programmée.
- Il traduit de façon cohérente le cahier des charges de l'automatisme.
- Il peut effectuer un choix rationnel des variables d'états et du codage du vecteur (ou mot) d'état.
- Il est bien adapté au cas des systèmes automatisés faisants intervenir un grand nombre de variables d'entrée.

Le GRAFCET est basé sur les notions d'étape et de réceptivité. Notons qu'un système automatisé évolue en passant par une succession d'étapes, aux quelles sont associées une ou plusieurs actions, et que le passage d'une étape à la suivante s'effectue en général lorsqu'une condition logique ou « réceptivité » est remplie. Le principe adopté dans un GRAFCET est de représenter l'automatisme par cet ensemble d'étapes auxquelles correspondront des « action », reliées entre elles per des « transitions », auxquelles correspondront des « réceptivités ».

Le mode de représentation qui est normalisé (Norme C03-190 de l'UTE), est le suivant (figure: I.20) :

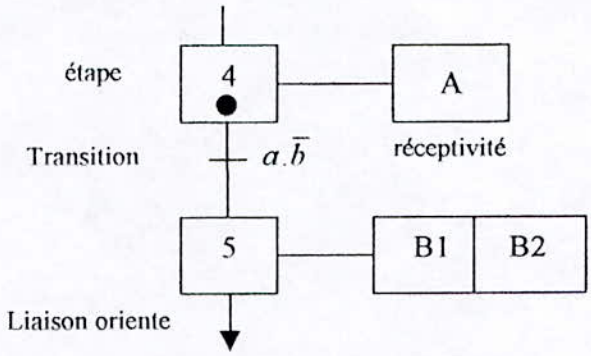


FIG.I.20: Représentation d'un GRAFCET

* Une étape est représentée par un carrée numéroté. Une étape active est indiquée par un point au dessous du numéro dans le carrée (par exemple, figure (I.20), l'étape 4 est active).

* Une « étape initiale » est représentée par un carrée double (en général c'est la situation de repos).

* Une « action » associée à une étape est représentée par un rectangle relié horizontalement au carrée correspondant.

* Une « transition »entre étapes est représentée par une barre perpendiculaire à la liaison orientée correspondante.

* Une « réceptivité » associée à une transition est inscrite à droite de la barre représentant la transition.

IV.3.3.2. Règles d'évolutions :

Pour obtenir un GRAFCET correct, il est nécessaire d'appliquer un certain nombre de règles fondamentales, exposées ci-dessous :

***Règle de syntaxe :**Alternance étape-transition :

L'alternance entre étapes et transitions doit être représentée quelle que soit la séquence en cours.

***Règles d'évolution :**

L'évolution de la situation d'un automatisme doit toujours satisfaire aux règles suivantes :

1.Situation initial :

Une situation initiale est caractérisée par le fait qu'un certain nombre d'étapes sont actives au début du fonctionnement (à l'initialisation).

2. Franchissement d'une étape :

Une transition entre étapes est dite « validée » si toutes ses étapes d'entrée sont actives. Elle sera franchie si elle est validée et si la réceptivité qui lui est associée est vraie: le franchissement est alors obligatoire et immédiat.

3. Evolution des étapes actives :

Le franchissement d'une transition entraîne l'activation de toutes les étapes immédiatement suivantes et la désactivation de toutes les étapes immédiatement précédentes.

4. Evolutions simultanées :

Plusieurs transitions simultanément franchissables sont simultanément franchies.

5. Activation et désactivation simultanées :

Si au cours du franchissement, une même étape doit être à la fois activée et désactivée, elle reste active.

Exemple d'application des règles 2 et 3 pour le franchissement d'une transition :

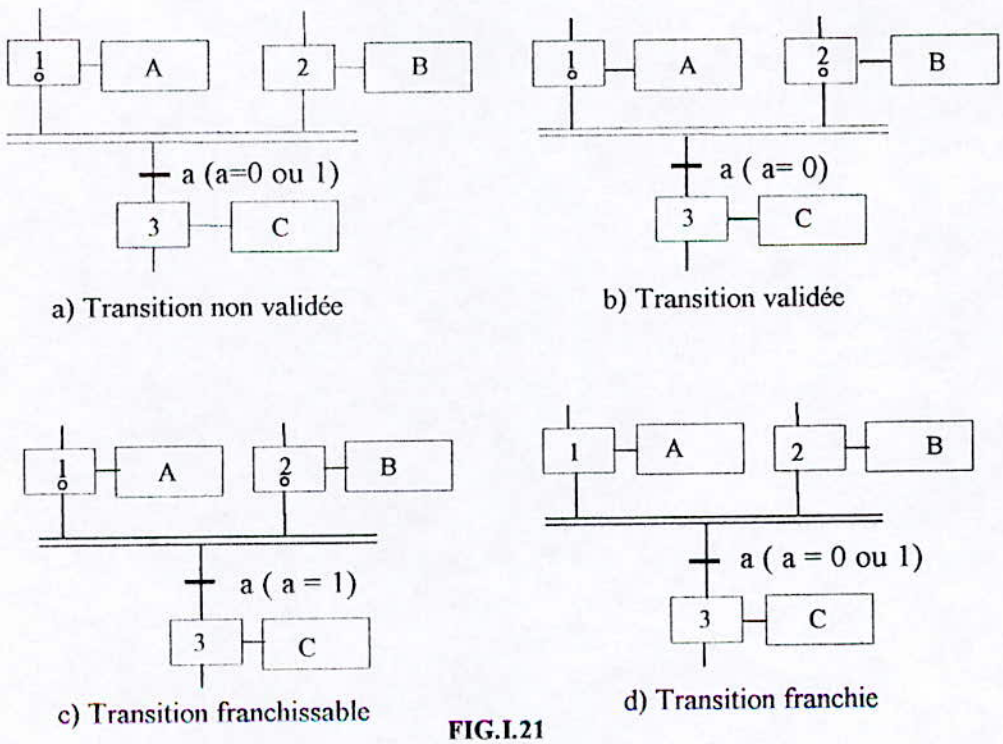


FIG.I.21

IV.3.3.3. Séquences simultanées :

Lorsque le franchissement d'une transition conduit à activer plusieurs étapes, les séquences de ces étapes sont dites « séquences simultanées » (ou aussi « séquences parallèles »). les séquences évoluent alors indépendamment les unes des autres ,et ce n'est que lorsque toutes les étapes finales de ces séquences sont actives simultanément, que l'évolution peut continuer sur une séquence unique par le franchissement simultané d'une même transition (figure : I.22).

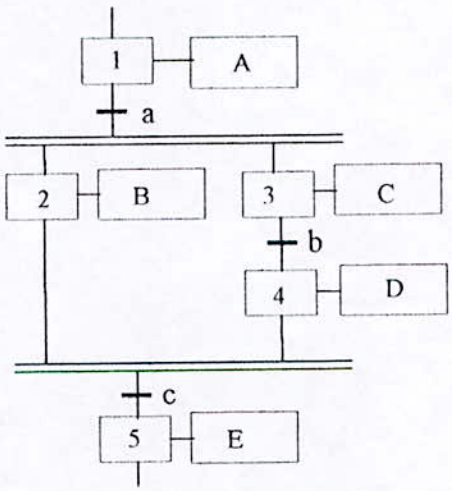


FIG.I.22

Le début et la fin de séquences simultanées doivent êtres représentées sur un grafcet par deux traits parallèles.

Par exemple (figure: I.22) lorsque l'étape 1est active et que la réceptivité a est vraie, les étapes 2 est 3sont activées simultanément.

Pour activer l'étape 5,il ne suffit pas que la réceptivité c soit vraie, il faut aussi que l'étape 4 soit activée ,ce qui se traduit pour avoir une réceptivité vraie de b.

Remarque :
Plusieurs séquences simultanées commencent toujours sur une même réceptivité unique et se terminent sur une réceptivité unique.

IV.3.3.4. Sélection des séquences:

Lorsque a partir d'une étape on peut effectuer un choix entre plusieurs évolutions sur des séquences débutants par des transitions dont les réceptivités sont exclusives, on a affaire a une « sélection de séquences » ou encore un «aiguillage».

Deux cas particuliers de sélection de séquence sont très utiles dans la pratique des systèmes séquentiels :

- **Le saut d'étapes:** qui permet de ne pas effectuer un certain nombre d'étapes lorsque celles ci sont inutiles .Par exemple (figure: I.23),à partir de l'étape 1, le système passera directement à l'étape 4 si la réceptivité a.b est vraie ,il évoluera normalement vers l'étape 2 si la condition a.b est vraie.

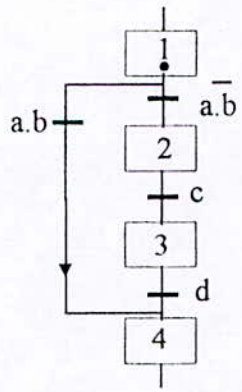


FIG.I.23

- **La reprise de séquence:** qui permet de recommencer une ou plusieurs fois la même séquence tant qu'une condition n'est pas réalisée(fig:I.24),à partir de l'étape 3, le système reprendra la séquence des étapes 2 et 3 tant que la condition a.b est réalisée, il évoluera normalement vers l'étape 4, si la condition a.b est vraie.

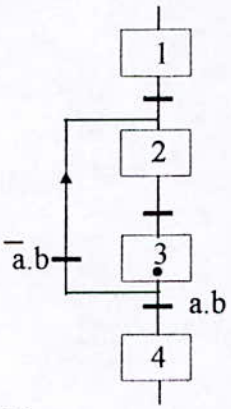


FIG.I.24

IV.3.3.5. ACTIONS PARTICULIERES :

L'analyse des actions associées à une étapes nécessite de bien faire la distinction entre la durée d'une action et la durée d'activation de l'étape correspondante. On convient habituellement de distinguer par « Xi » l'état actif de l'étape « i », exemples:

X3 = 1 si l'étape 3 est active.

X5 = 1 si l'étape 5 est active.

1. Action continue:

Le cas le plus simple est celui d'une action continue, qui se poursuit tant que l'étape à la quelle est associée reste active.

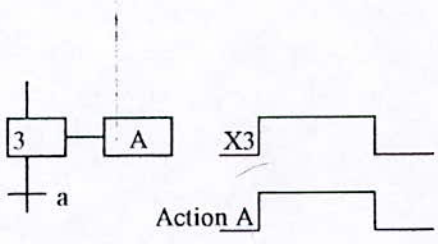


FIG.I.25

Par exemple (figure:I.25), l'action A associée à l'étape 3 dure tant que cette étape est active.

2. Action conditionnelle :

Une action conditionnelle est une action continue dont l'exécution est soumise à la réalisation d'une condition logique U . Cette condition est alors notée à coté d'un tiret dessiné à la partie supérieure du rectangle qui représente l'action (figure:1.26).

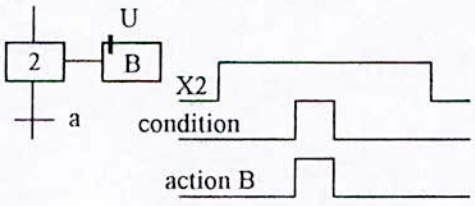


FIG.I.26

3. Action temporisée :

Une action temporisée est une action conditionnelle dans la quelle le temps intervient comme condition logique. La notation générale adoptée pour faire intervenir le temps est « t/i/q sec », ou i désigne le numéro de l'étape comportant l'action de comptage du temps, et q est la durée écoulée depuis l'action de l'étape i.

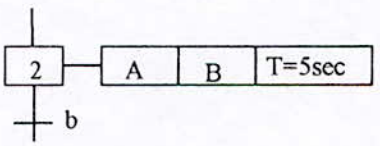


FIG.I.27

Par exemple (figure:I.27), deux actions temporisée A et B sont associées à l'étape 3: l'action A sera exécutée si 2 secondes sont déjà écoulées depuis le début de l'activation de l'étape 1, et si l'étape 3 est active.

L'action B sera exécutée si 5 secondes ne sont pas encore écoulées depuis le début de l'activation de l'étape 1, et si l'étape 3 est active.

4. Action de comptage d'un temps : On représente ce type de d'actions habituellement par la notation «T= q sec », à l'intérieure du rectangle figurant l'action (q est la durée à compter) . Par exemple (figure:I.28) trois actions sont associées à l'étape 2 : A, B et le comptage de 5sec (plus exactement l'initialisation du comptage de 5sec).

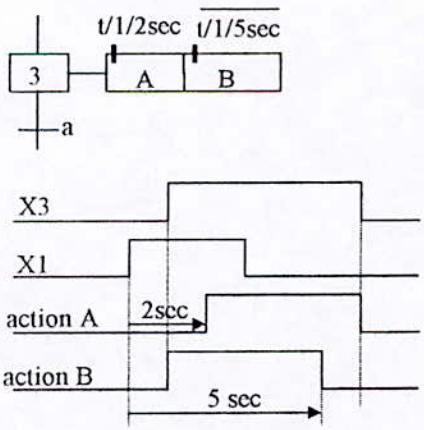


FIG.I.28

IV.3.3.6. RECEPTIVITES PARTICULIERES :

Les réceptivités associées à une transition peuvent être des formes particulières, dont deux sont assez fréquemment rencontrées : les réceptivités fonctions du temps et les réceptivités caractérisant un changement d'état (et non pas un état seulement).

1. Réceptivité fonction de temps:

La réceptivité d'une transition peut être constituée par la constatation d'une durée écoulée depuis le début de l'activation d'une étape comprenant elle-même le comptage d'un temps. La notation utilisée est la même que celle déjà utilisée.

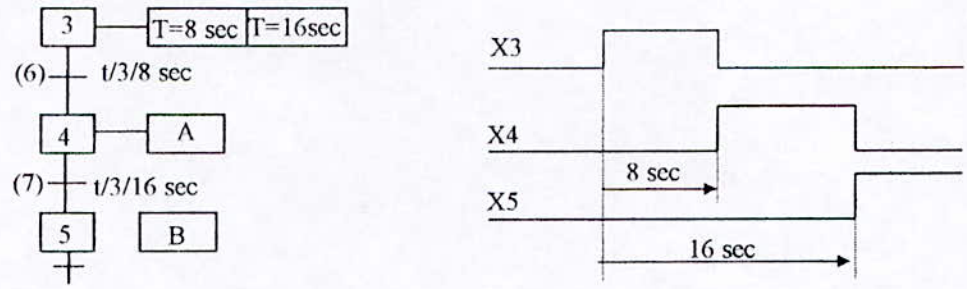


FIG.I.29: Réceptivités fonction du temps.

Par exemple (figure:I.29), deux initialisations de comptages des temps T= 8sec et T= 16sec sont associées à l'étape 3. La réceptivité de la transition 6 sera validée 8sec parés l'activation de l'étape 3, et la réceptivité de la transition 7 sera validée 16sec parés l'activation de cette même étape.

2. Réceptivité faisant intervenir un changement d'état :

Il est fréquent que l'on ait, à détecter dans un automatisme, le changement d'état d'une variable (capteur de fin de course à impulsion, front montant ou descendant de tel ou tel « bit » électronique d'un registre d'état, etc.....).

On présente habituellement par la notation « $\uparrow a$ » le passage d'une variable a de l'état logique a = 0 à l'état logique a = 1 (front montant), et par la notation « $\downarrow a$ » le passage de l'état logique a = 1 à l'état logique a = 0 (front descendant).

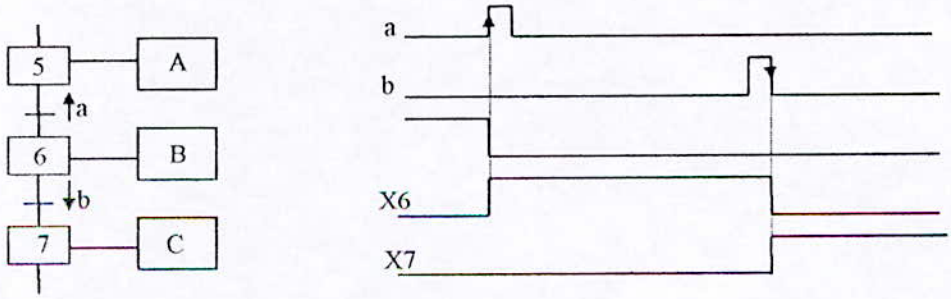


FIG.I.30: Réceptivité faisant intervenir un changement d'état.

La réceptivité d'une transition peut être constituée par le changement de l'état d'une variable. Par exemple (figure:I.30), l'étape 6 devient active à l'instant du front montant de la variable a, et l'étape 7 deviendra active à l'instant du front descendant de la variable b.

On peut remarquer qu'il est toujours possible de remplacer une seule transition dont la réceptivité fait intervenir un changement d 'état d'une variable par deux transitions successives dont les réceptivités ne font intervenir que les états statiques de cette variable.

Ainsi les deux séquences de (figure:I.31) sont équivalentes : la transition entre les étapes 2 et 3 contrôle l'absence de a par la réceptivité, et la transition entre les étapes 3 et 4 contrôle la présence de a par la réceptivité a.

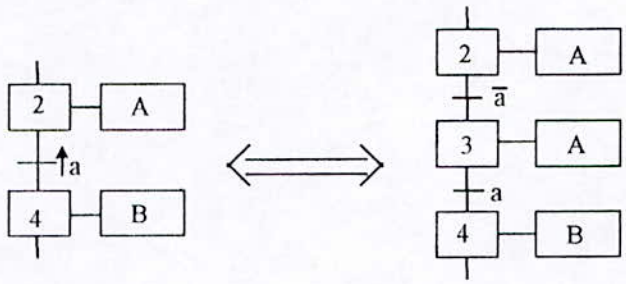
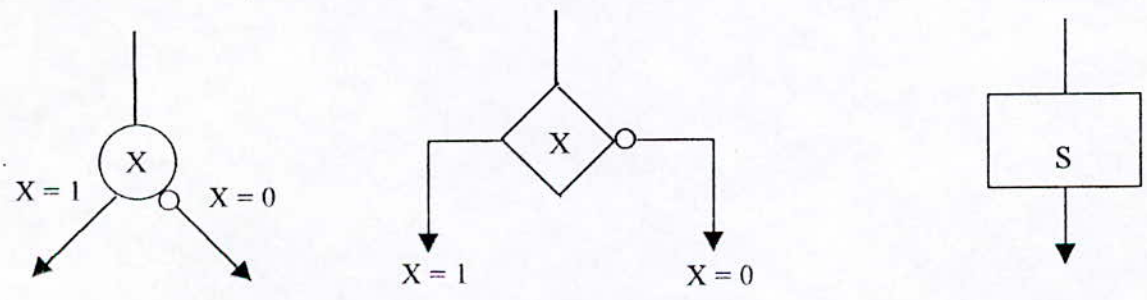


FIG.I.31: Remplacement de la réceptivité ↑ a.

IV.3.4. Graphe de décision binaire :

Le graphe de décision binaire est constitué par l'assemblage de deux types d'éléments * Un élément de test, représenté par un losange ou un nœud (fig. I.32-a) et défini par une variable de test logique X chaque élément de ce type possède une borne d'entrée, deux bornes de sortie : X = 1 et X = 0; cette dernière borne est mise en évidence par un petit rond, symbole général de l'inversion logique ;

*Un élément d'affectation (appelé aussi élément d'action) représenté par un rectangle (fig.I.32-b) et défini par une affectation S ; chaque élément de ce type possède une borne d'entrée et une borne de sortie unique.



a) b)

FIG I.32

A chaque période du signal d'horloge, un système séquentiel synchrone lise et exécute un élément de test ou un élément d'affectation du diagramme. L'exécution d'un élément de test détermine, selon la valeur de la variable logique, le prochain élément à exécuter, tandis que l'exécution d'un élément d'affectation produit un état de sortie. Les deux éléments de test et d'affectation constituent les deux primitives des spécifications.

IV.4. Description par microprogramme :

VI.4.1 : Définition :

Tout assemblage d'opérateurs logiques combinatoires (inverseur, portes ET, OU, NAND, MUX, DMUX,...etc) et séquentiels (verrou, bascule, registre,...etc. Mais à l'exclusion des mémoires) constitue un système logique câblé et d'une (ou plusieurs) mémoire (s) est un système logique à mémoire (s). Lorsque le contenu de cette (ces) mémoires est celui des code des primitives des spécifications (éléments de test et d'affectation), on dit qu'il s'agit d'un système logique microprogrammée et ce contenu est le microprogramme.

IV.4.2. Méthodes :

La réalisation logicielle des diagrammes de décision binaire nécessite les étapes suivantes :

- définition détaillée des primitives et réalisations de celles-ci en mémoire sous la forme de micro-instructions;
- représentation de l'arbre ou du diagramme par un assemblage de micro-instructions;
- traduction de l'organigramme en un micro programme mnémonique ,puis conversion de celui-ci en un microprogramme binaire ou hexadécimal;
- réalisation matérielle du système séquentiel exécutant le microprogramme.

IV.4.3. Micro-instructions de test et d'affectation :

Chaque élément de test et d'affectation doit être codé en binaire, écrit en mémoire à une adresse donnée et, en fin, exécuté par une machine séquentielle ad hoc. La réalisation d'un élément de test ou d'affectation dans une mémoire produit une micro-instruction de test ou une micro-instruction d'affectation.

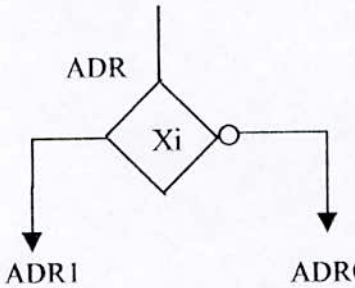
Une micro-instruction de test (figure(I.33-a)) est définie par une adresse ADR et une variable des test Xi ; chaque micro-instruction suivante d'adresse ADR1(Xi=1) ou d'adresse ADR0 (Xi=0). La micro-instruction de test est également décrite par une écriture symbolique mnémonique :

IF Xi **Then** ADR1 **ELSE** ADR0
Si Xi **Alors** ADR1 **Si non** ADR0

Une micro-instruction d'affectation (figure(I.33-b)) est définie par une adresse ADR et une affectation synchrone REG← OUT où REG désigne un registre et OUT un état de sortie ; chaque micro-instruction de ce type possède une borne d'entrée et une borne de sortie conduisant à la micro-instruction suivante d'adresse ADR+; l'expression mnémonique correspondante s'écrit sous la forme :

DO REG← OUT **GOTO** ADR+
(**Faire** REG ← OUT **puis** ADR+)

On remarque que, contrairement à élément d'affectation, la micro-instruction d'affectation possède une borne de sortie.



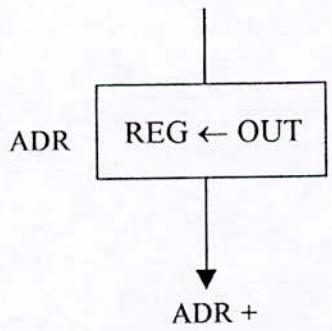


Fig.I.33 : **a)** micro-instruction de test, **b)** micro-instruction d'affectation

VI.4.4. Micro-instruction d'affectation avec indice :

Dans de nombreuses applications, on veut disposer de plusieurs registres d'affectation distincts; l'instruction d'affectation prend la forme plus générale.

DO REG [j]← OUT **GOTO** ADR+
(faire REG[j]← OUT **puis** ADR+)

où j est un indice ou adresse distinguant un registre parmi j.

IV.4.5. Organigramme :

Un assemblage quelconque de micro-instruction de test et d'affectation peut être représenté graphiquement à l'aide des symboles des figures (I.33-a) et (I.33-b) ; on dira qu'une

telle représentation est un organigramme ou schéma de programme. Notons que si tout diagramme de décision binaire est un organigramme, l'inverse n'est pas toujours vrai.

Le diagramme de décision binaire ou l'organigramme de l'exemple 2 est le suivant:

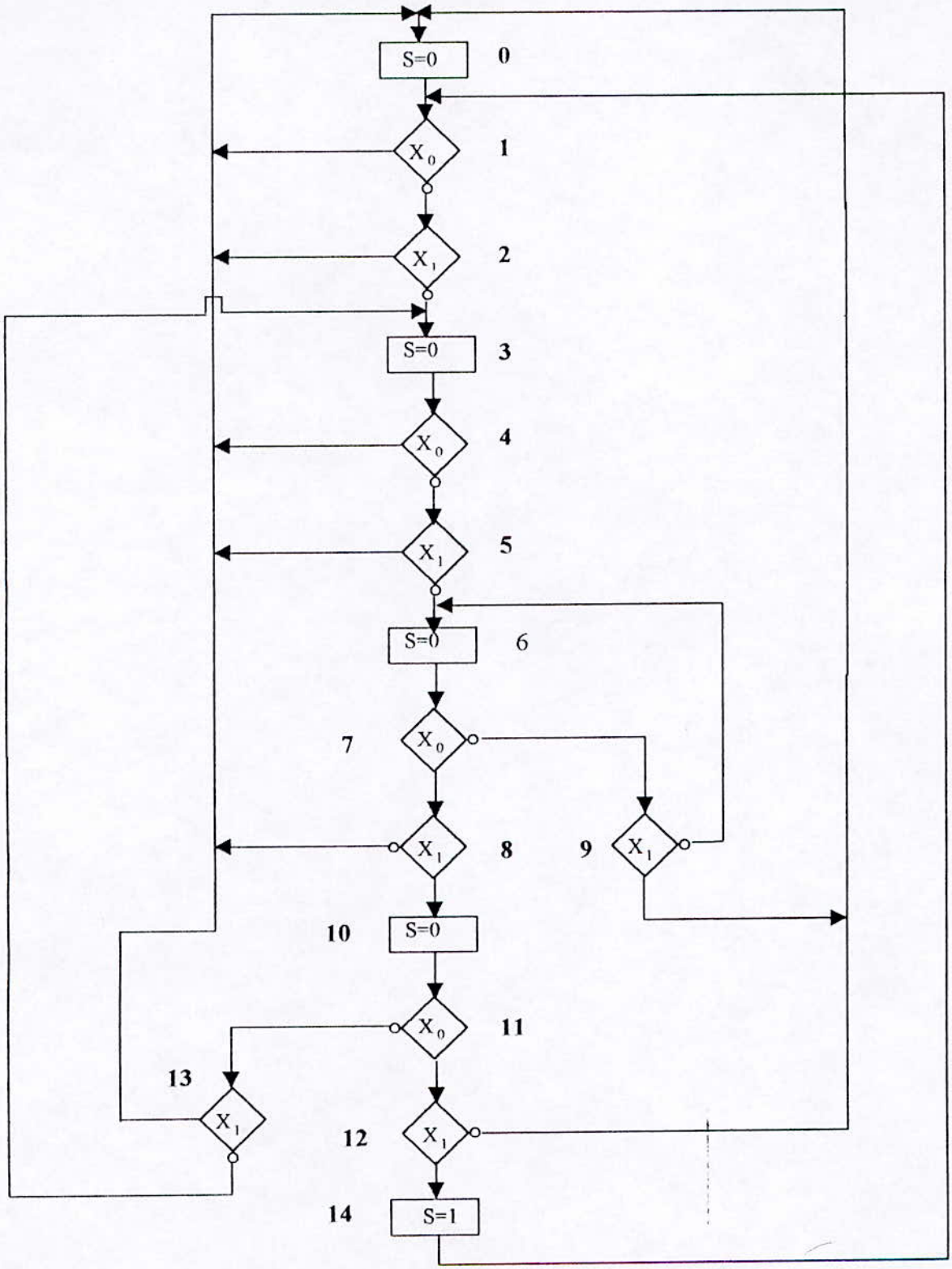


FIG.I.34 : Diagramme de décision binaire de l'exemple 2.

IV.4..6. Microprogramme mnémonique :

Pour cette section on va considérer seulement les microprogrammes linéaires sans compteur ordinal. Les microprogrammes non linéaires avec compteur ordinal seront explicités au deuxième chapitre.

Après avoir affecté à chaque micro-instruction de l'organigramme une adresse, il est possible de dresser une liste de tout les micro-instructions, exprimées chacune à l'aide de leur expression mnémonique ; une telle liste constitue le microprogramme mnémonique. Le microprogramme mnémonique de l'exemple 2 sera le suivant :

ADR	Microprogramme mnémonique
0	**DO** REG ← 0 **GOTO** 1
1	**IF** X_0 = 0 **THEN** 2 **ELSE** 0
2	**IF** X_1 = 0 **THEN** 2 **ELSE** 0
3	**DO** REG ← 0 **GOTO** 4
4	**IF** X_0 = 0 **THEN** 5 **ELSE** 0
5	**IF** X_1 = 0 **THEN** 6 **ELSE** 0
6	**DO** REG ← 0 **GOTO** 7
7	**IF** X_0 = 0 **THEN** 9 **ELSE** 8
8	**IF** X_1 = 0 **THEN** 0 **ELSE** 10
9	**IF** X_1 = 0 **THEN** 6 **ELSE** 0
10	**DO** REG ← 0 **GOTO** 11
11	**IF** X_0 = 0 **THEN** 13 **ELSE** 12
12	**IF** X_1 = 0 **THEN** 0 **ELSE** 14
13	**IF** X_1 = 0 **THEN** 3 **ELSE** 8
14	**DO** REG ← 1 **GOTO** 1

FIG.I.35 : microprogramme mnémonique de l'exemple 2.

IV.3.7: microprogrammes binaire et hexadécimal :

L'écriture des micro-instruction dans la mémoire déroule :

* d'un codage des variables de test Xi et des affectations ;

* d'une organisation précise des divers éléments de la micro-instruction : c'est le format de la micro-instruction, lui-même décomposé en plusieurs champs.

* d'un codage de toutes les adresses.

* et enfin la distinction des deux micro-instructions nécessite une information supplémentaire, un seul bit T (pour un codage minimale) suffit à distinguer une micro-instruction de test (T=1) d'une micro-instruction d'affectation (T=0). Aussi, pour distinguer entre le test par rapport à zéro et le test par rapport à un on doit ajouter une information présentée par un seul bit i_0 :

i_0= 1 pour le test par rapport à1 , i_0=0 pour le test par rapport à 0.

Après avoir codé les variables de test, les affectations, les adresses et les micro-instructions, il est possible de proposer le format de la figure (I.37) et rédiger le microprogramme en binaire. L'exécution des deux micro-instructions des figures (I.33-a,I.33-b) par une machine spécialisée impose des largeurs identiques et des contraintes sur les emplacements respectifs de divers champs: code opératoire et opérande.

* **code opératoire :** contient le code des micro-instructions de test et d'affectation ainsi que celui des variables d 'entrée et le bit de distinction des test par rapport à zéro ou à un.

* **opérande :** contient le code des adresses et la sortie. L'emplacement de ses champs d'adresse et de sortie doit être bien spécifier, ainsi les adresses ADR1 et ADR+ auront la même position en mémoire et l'adresse ADR avec la sortie OUT occuperont la même position aussi, il en déroule que certains bits marqués par des tirets(-), sont indéterminés.

Pour des commodités d'écriture, on convertit fréquemment le microprogramme hexadécimal équivalent ; cette conversion s'effectuera en supposant que tous les bits indéterminés (-) seront égaux à 0 (ou à 1).

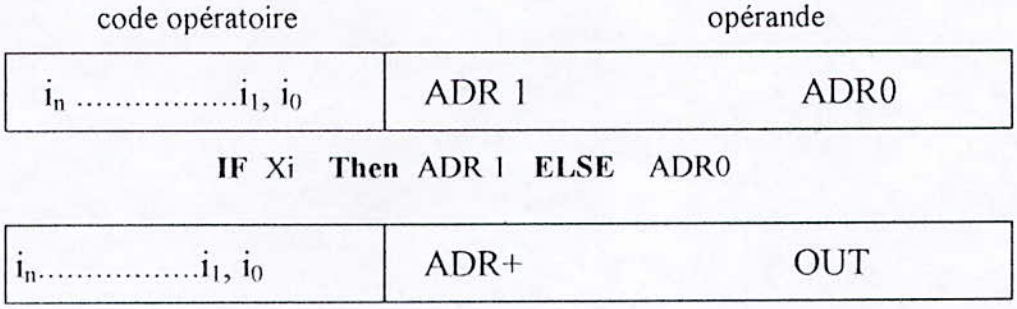


FIG.I.36 : format des micro-instructions.

Le microprogramme en binaire de l'exemple précèdent sera ainsi : résultat d'un codage des différents variables en respectant le format imposé :

on a 2 variables d'entrée, Xo, X1 donc on a besoin d'un seul bit et un autre pour la sortie; par contre on a 15 adresses donc 4 bits pour les représenter en code minimal. Un bit pour coder les constructions de teste et d'affectation et un bit pour distinguer les deux test : test par rapport à zéro et un autre par rapport à 1.

$X_0 \rightarrow i_1 = 0$ DO $\rightarrow i_2 = 0$ IF $X_i = 0 \rightarrow i_0 = 0$

$X_1 \rightarrow i_1 = 1$ IF $\rightarrow i_2 = 1$ IF $X_i = 1 \rightarrow i_0 = 1$

ADR	i2 i1 i0	ADR1 ADR+	ADR0 OUT
0 0 0 0	0 - -	0001	- - - 0
0 0 0 1	1 0 0	0010	000 0
0 0 1 0	1 1 0	0011	000 0
0 0 1 1	0 - -	0100	- - - 0
0 1 0 0	1 0 0	0101	000 0
0 1 0 1	1 1 0	0110	000 0
0 1 1 0	0 - -	0111	- - - 0
0 1 1 1	1 0 0	1001	100 0
1 0 0 0	1 1 0	0000	101 0
1 0 0 1	1 1 0	0110	000 0
1 0 1 0	0 - -	1011	- - - 0
1 0 1 1	1 0 0	1101	110 0
1 1 0 0	1 1 0	0000	111 0
1 1 0 1	1 1 0	0011	100 0
1 1 1 0	0 0 0	0001	- - - 1

FIG.I.37 : microprogramme en binaire

Le microprogramme hexadécimal correspondant est le suivant :

(les bits indéterminés sont tous mises à zéro).

ADR	OPC	Operande	
0	0	1	0
1	4	2	0
2	6	3	0
3	0	4	0
4	4	5	0
5	6	6	0
6	0	7	0
7	4	9	8
8	6	0	A
9	6	6	0
A	0	B	0
B	4	D	C
C	6	0	E
D	6	3	8
E	0	1	1

FIG.I.38 : microprogramme en hexadécimal.

VI - Conclusion :

Une fonction logique ou discrète peut toujours être représentée par un diagramme de décision binaire. Ce diagramme, après éventuelle simplification, peut être directement réalisé par un système microprogramme (réalisation logicielle ou par un système câblé (réalisation matérielle). Le système microprogrammé effectue une évolution temporelle (ou séquentielle) des diagrammes et nécessite un certain matériel (la machine de décision binaire). Une réalisation logicielle exige de temps te économise de l'espace (ou matériel), contrairement à une réalisation matérielle qui gagne du temps au prix d'un certain espace.

En outre, les microprogrammes ont évalués par un paramètre spatial, défini par le nombre d'instruction ou coût de l'espace en mémoire.

CHAPITRE II :

Machines sequentielles microprogrammées

CHAPITRE II : MACHINES SEQUENTIELLES MICROPROGRAMMEES :

I- Introduction :

Nous avons vu au premier chapitre les méthodes de description des machines séquentielles. La méthode microprogrammée est la plus efficace pour des systèmes plus complexes, en effet, elle nous permet de jouer sur le *soft* et sur le *hard* au même temps pour avoir un compromis entre les deux.

En se trouve parfois à des situations où le microprogramme présente des blocs d'instructions qui se répètent souvent dans le même microprogramme qui, lui même, occupe un espace mémoire très considérable.

L'introduction des nouvelles notions sera très nécessaire pour éliminer toutes les séquences (opérations) redondantes ou répétitives. Un sous microprogramme appelé par des nouvelles instructions sera désigné par procédure, cette dernière nécessite une solution matérielle peu coûteuse mais qui introduira un gain considérable en logiciel .

Le format des instructions du premier chapitre nécessitait la spécification de deux adresses . L'introduction d'un compteur ordinal donne lieu à un nouveau concept de microprogramme : microprogramme incrementé, ce dernier ne nécessite que la spécification d'une seule adresse, l'autre adresse sera l'adresse présente incrementée de un.

II- Machine à microprogramme linéaire :

Un microprogramme linéaire est un microprogramme qui peut être réalisé directement du graphe des états (ou table des transitions) à l'aide des deux instructions de test et d'affectation, déjà définies au premier chapitre, le microprogramme final sera réduit à un seul microprogramme principale.

Exemple: soit un système séquentiel conçu pour comparer les sorties de deux sources binaires X1 et X2 bit par bit, et qui délivre une brève impulsion à sa sortie S dès qu'il compte 4 bits égaux.

La table des états d'un tel système sera comme suit: (figure.II.1)

état présent	états futurs x_1 x_2				sortie
	00	01	10	11	S
0	1	0	0	1	0
1	2	1	1	2	0
2	3	2	2	3	0
3	4	3	3	4	1
4	0	0	0	0	0

FIG.II.1 : table des états pour l'exemple du comparateur.

Le diagramme de décision binaire correspondant est un microprogramme linéaire (figure.II.2)

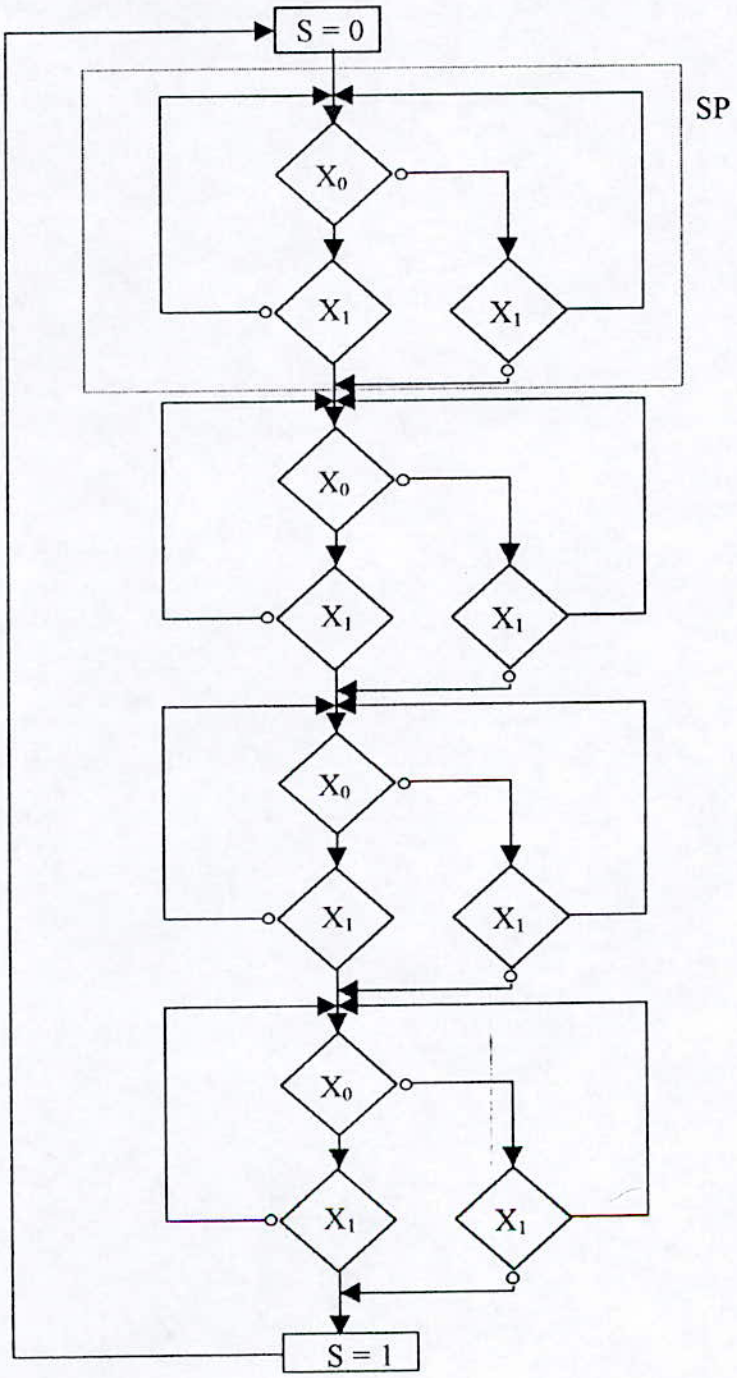


FIG.II.2 : DDB du comparateur.

III- Machine à microprogramme non linéaire :

Un microprogramme non linéaire est un microprogramme qui fait exploiter les propriétés de redondance des blocs d'instructions de test et d'affectation d'un microprogramme au moyen de nouvelles instructions mnémoniques.

III.1- Sous microprogramme :

Tout organigramme comportant une borne d'entrée unique et une borne de sortie unique représente un microprogramme particulier appelé sous microprogramme. Le bloc en pointillé du diagramme de la figure (II.2) représente un sous microprogramme.

L'utilité des sous microprogrammes est de simplifier les microprogrammes finals . Un sous microprogramme répétitif peut être transformer en un seul microprogramme .L'appel à ce dernier ce fait chaque fois que l'exécution du microprogramme le nécessite. Cette tâche est effectuée par une réalisation matérielle « la pile ». L'appel au sous microprogramme par des nouvelles instructions mnémoniques le transforme en une procédure.

III.2- Procédure et pile :

III.2.1- Définition : Pile :

La pile est un registre particulier (en anglais push down stak register ou LIFO register : last -in - first - out - register) qui peut enregistrer, au moment de quitter un état présent, l'adresse de l'état futur qui sera atteinte au terme de l'exécution du sous microprogramme. La présence de cet opérateur matériel, la pile, entraîne des exigences logicielles et, en particulier, l'existence de deux nouvelles instructions : instruction d'appel de sous microprogramme et l'instruction de retour au microprogramme principal.

III.2.2- Micro-instructions d'appel et de retour :

Une micro-instruction d'appel est définie par une adresse ADR et par l'adresse ADSP de la première instruction du sous microprogramme; graphiquement, chaque instruction de ce type possède une borne d'entrée et une borne de sortie conduisant à l'adresse de retour ADRT (figure II.3) son expression mnémonique s'écrit sous la forme :

CALL ADSP **RET TO** ADRT

Une instruction de retour (figure II.4) est définie par son adresse ADR et par la simple expression mnémonique : RET. Chaque instruction de ce type possède une unique borne d'entrée

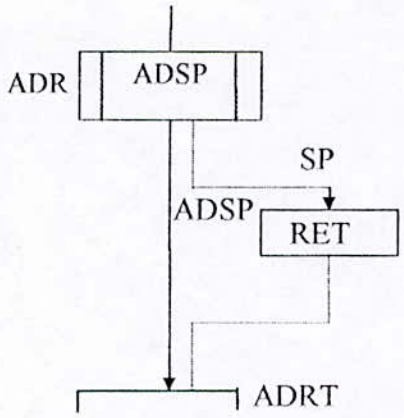


FIG.II.3: micro-instruction d'appel.

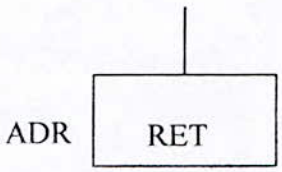


Fig.II.4: micro-instruction de retour.

III.2.3- Procédure : On appelle procédure tous sous microprogramme appelé par une instruction d'appel qui appartient à un microprogramme principal ou une autre procédure. Une procédure est toujours représentée par un organigramme comportant une borne d 'entrée unique, aucune borne de sortie, et une instruction terminale : l'instruction de retour.

***remarque** : Quand l'instruction d'appel appartient à la procédure elle-même, on dira que celle-ci est récursive .*

En utilisant les symboles graphiques des figures (II.3) (où les traits mixtes sont négligés) et (II.4) , on transforme le microprogramme de la figure(II.2) en un microprogramme principal et un sous microprogramme en conservant une copie unique du sous microprogramme. Figure (II.5).

III.2.4- Chronologie d'exécution :

* à l'adresse initiale 0, exécution de l'affectation Z ← 0 ;

* à l'adresse ADR =1, appel du sous microprogramme débutant à l'adresse ADSP = 6, et pour la mise en mémoire, dans la pile, de l'adresse de retour du sous microprogramme ADRT = 2 ;

* exécution séquentielle des instructions d'adresse 6 , 7 et 8 du sous microprogramme ;

* à l'adresse 9, restitution de l'adresse ADRT =2 stockée dans la pile par l'instruction RET et exécution du retour du sous microprogramme au microprogramme principal, à cette dernière adresse ;

* à l'adresse ADRT = 2, exécution de l'appel une seconde fois au sous microprogramme.

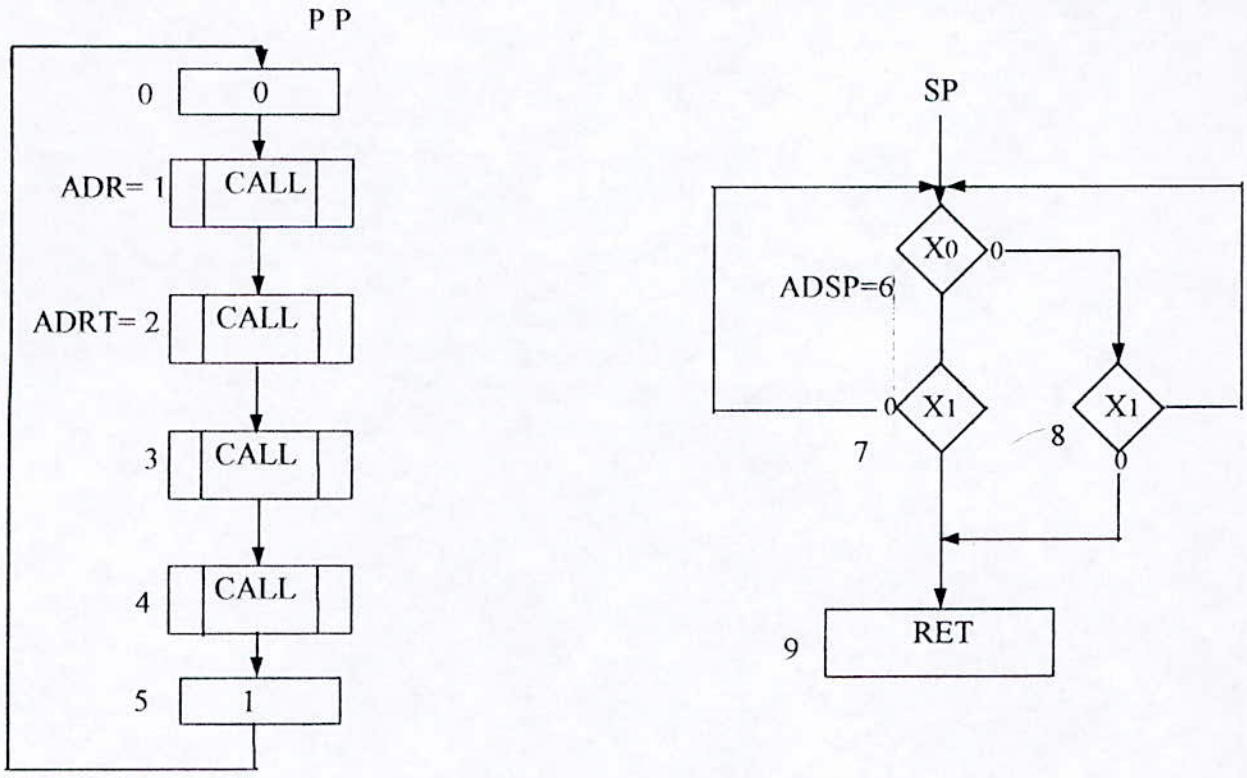


FIG.II.5 : DDB du l'exemple précédent avec procédure.

Le microprogramme mnémonique correspondant est le suivant : figure (II.6)

ADR	MICROPROGRAMME MNEMONIQUE
0	**DO** REG ← 0 **GOTO** 1
1	**CALL** 6 **RET TO** 2
2	**CALL** 6 **RET TO** 3
3	**CALL** 6 **RET TO** 4
4	**CALL** 6 **RET TO** 5
5	**DO** REG ← 1 **GOTO** 0
6	**IF** $X_0 = 1$ **THEN** 7 **ELSE** 8
7	**IF** $X_1 = 1$ **THEN** 9 **ELSE** 6
8	**IF** $X_1 = 1$ **THEN** 6 **ELSE** 9
9	**RET**

FIG.II.6 : microprogramme mnémonique du comparateur.

Le format des micro-instruction se compose du code opératoire et l'opérande : Le code opératoire contient le codage des quatre micro-instructions qui nécessite deux bits i_3 i_2, un seul bit i_1 suffit pour le codage des deux variables d'entrée X0 , X1 et un autre bit i0 pour la distinction entre le test par rapport à zéro ou par rapport à 1.

instructions	D0	IF	CALL	RET
codes	00	01	10	11

* $X0 \rightarrow i_1 = 0$

  $X1 \rightarrow i_1 = 1$

* Si le test est de la forme « IF Xi = 1 THEN... » alors i0 = 1;

  Si par contre il est de la forme « IF Xi = 0 THEN ... » alors i0 = 0.

L'opérande contient les deux adresses des instructions de test et d'affectation, l'adresse futur et la sortie de l'instruction d'affectation et des bits à des valeurs indifférentes pour l'instruction de retour.

Le format général se présente sous la forme :

i_3 i_2 i_1 i_0	ADR1 ADR0 ADR+ OUT ADSP ADRT

code opératoire opérande

On a deux états de sortie donc il nous faut un seul bit pour un code minimal et 10 adresses, ce qui nécessite 4 bits.

Le microprogramme en binaire est représenté dans la figure (II.7) et le microprogramme en hexadécimal correspondant est représenté dans la figure (II.8).

ADR	i_3 i_2 i_1 i_0	ADR1 ADR+ ADSP	ADR0 OUT ADRT
0 0 0 0	0 0 - -	0 0 0 1	- - - 0
0 0 0 1	1 0 - -	0 1 1 0	0 0 1 0
0 0 1 0	1 0 - -	0 1 1 0	0 1 0 0
0 0 1 1	1 0 - -	0 1 1 0	0 1 0 0
0 1 0 0	1 0 - -	0 1 1 0	- - - 1
0 1 0 1	0 0 - -	0 0 0 0	1 0 0 0
0 1 1 0	0 1 0 1	0 1 1 1	0 1 1 0
0 1 1 1	0 1 1 1	1 0 0 1	0 1 1 0
1 0 0 0	0 1 1 1	0 1 1 0	1 0 0 1
1 0 0 1	1 1 - -	- - - -	- - - -

FIG.II.7 : microprogramme en binaire.

ADR	ROM [ADR]
0	0 1 0
1	8 6 2
2	8 6 3
3	8 6 4
4	8 6 5
5	0 0 1
6	5 7 8
7	7 9 6
8	7 6 9
9	9 0 0

FIG.II.8 : microprogramme en hexadécimal.

(toutes les valeurs indifférentes sont mises à 0).

III.2.5- Description de la pile :

III.2.5.1- Registre à décalage bidirectionnel :

Un registre à décalage bidirectionnel de m bits est un assemblage de m bascules et de m multiplexeurs selon la configuration de la figure (II.9) qui représente un registre à décalage à 4 bits.

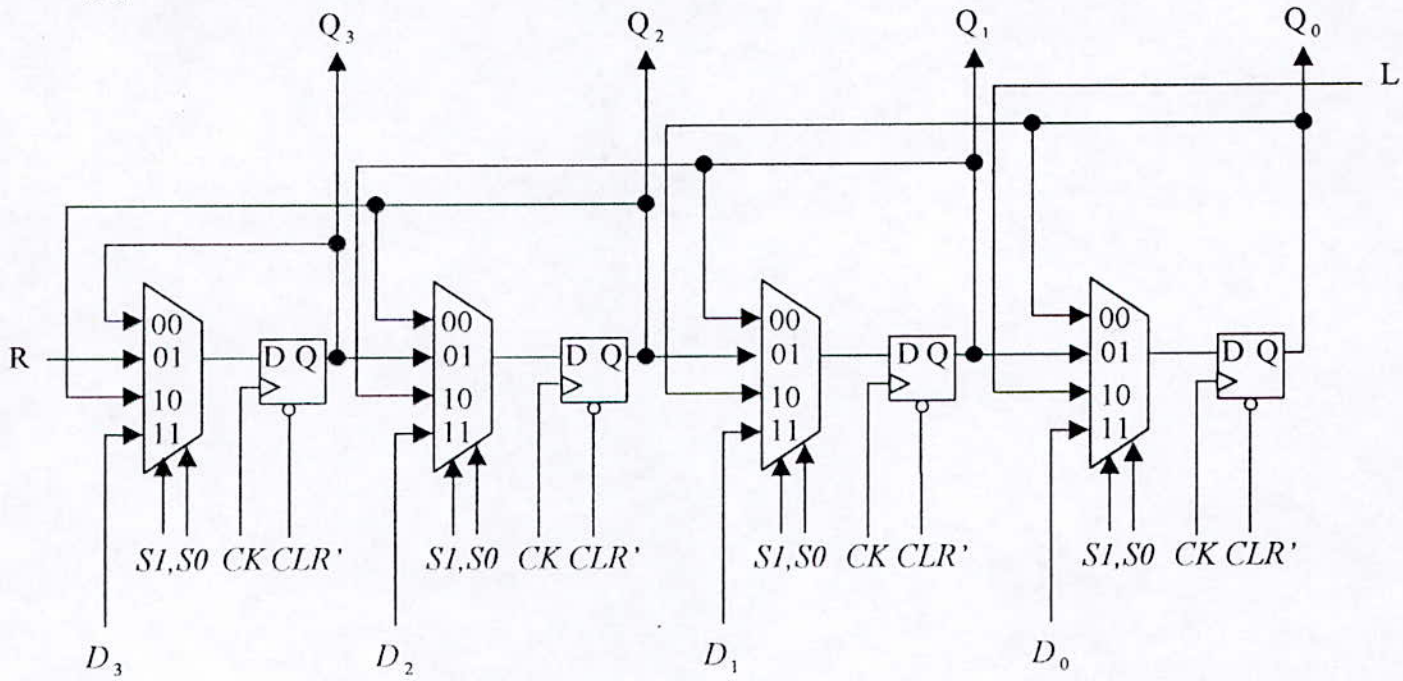


FIG.II.9 : registre à décalage bidirectionnel de 4 bits.

On peut représenter cette figure d'une manière plus compact comme suit :figure (II.10)

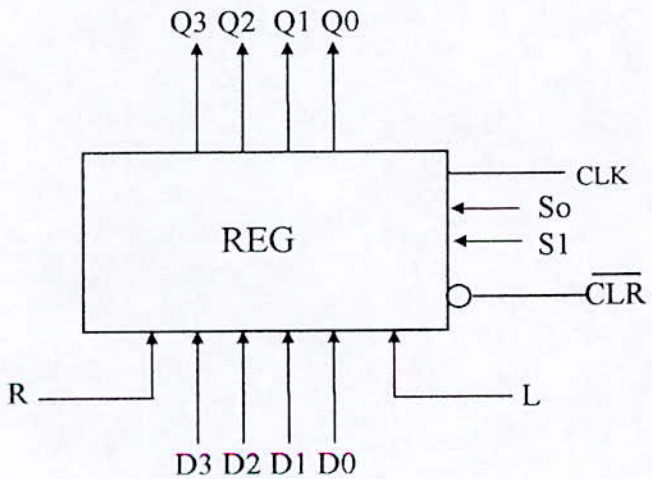


FIG.II.10 : logigramme d'un registre à décalage.

La table des opérations d'un tel registre est la suivante :

opération	description	CLR	S1	S0
CLEAR	Q = 0	0	∅	∅
HOLD	Q ← Q	1	0	0
SHIFT RIGHT	Q ← Q/2, Q3← R	1	0	1
SHIFT LEFT	Q ← Q *2, Qo← L	1	1	0
LOAD	Q ← D	1	1	1

FIG.II.11 : table des opérations d'un registre à décalage bidirectionnel.

III.2.5.2-Pile :

Une pile à m niveaux pour mots de un bit, est un registre bidirectionnel de m bits comportant m bascules D .

Le logigramme de la figure (II.12.-a-) et le schéma de la figure (II.12-b-) représentent une pile à quatre niveaux (Qo... Q3), pour mots de un bits (IN est l'entrée et OUT = Qo est la sortie) ; construite avec le registre de la figure (II.9) le mot supérieur (Q0 = OUT) est le sommet de la pile, le mot inférieur (Q3) en est le fond .

En négligeant l'influence de l'entrée asynchrone CLR , la table des opérations de la figure (II.13) comporte quatre opérations :

* Une opération neutre (HOLD) pour PUSH, POP = 00 ;
* Une opération de dépilage (POP) pour PUSH, POP =01 ; qui entraîne un décalage d'un bit du bat vers le haut de la pile ;
* Une opération d'empilage (PUSH) pour PUSH, POP = 10 ; qui effectue un décalage d'un bit du haut vers le bat de la pile ;
* Une opération de chargement (LOAD) pour PUSH, POP =11.

Le registre bidirectionnel se transforme en pile si l'on vérifie les relations suivantes :

S1 = PUSH ; S0 = POP

R = ∅ (SHIFT RIGHT = POP)

L = IN (SHIFT LEFT = PUSH)

D3 = Q3 , D2 = Q2 , D1 = Q1 , D0 = IN (LOAD).

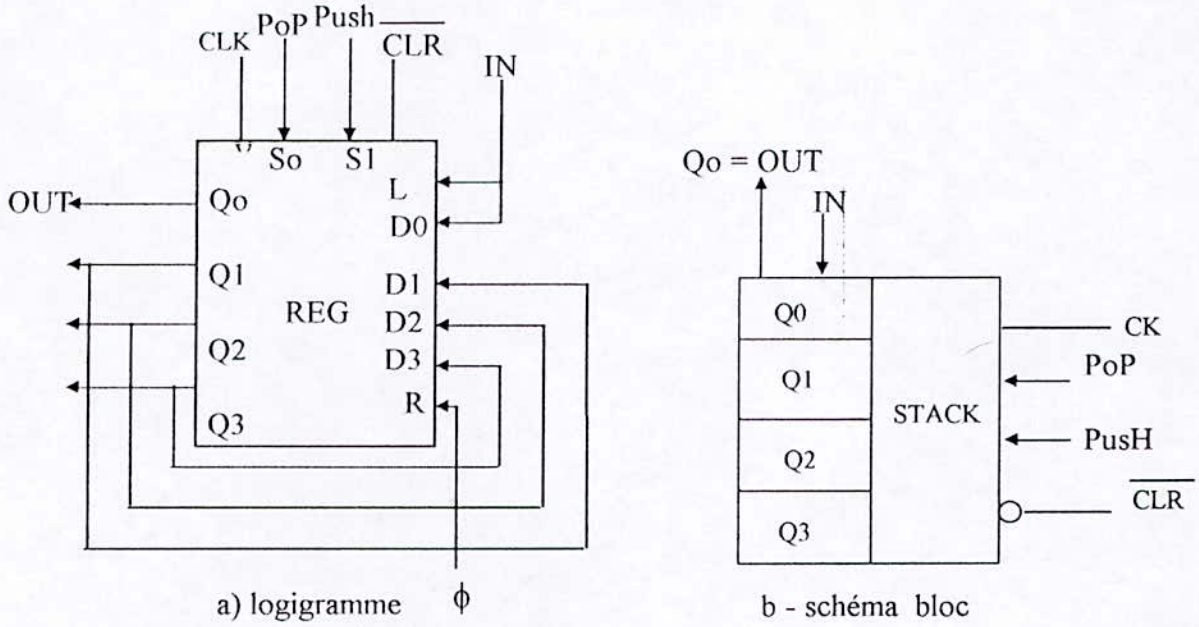


FIG.II.11 : logigramme et schémas bloc d'une pile.

opération	Description	PUSH = S1	POP = S0
HOLD	Q3←Q3, Q2←Q2, Q1← Q1, Q0←Q0	0	0
POP	Q3←Q, Q2←Q3, Q1←Q2, Q0←Q1	0	1
PUSH	Q3←Q2, Q2←Q1, Q1←Q0, Q0←IN	1	0
LOAD	Q3←Q3, Q2←Q2, Q1←Q1, Q0←IN	1	1

FIG.II.12: table des opérations d'une pile.

Une pile à un m niveau , pour mots de n bits , est un assemblage itératif de n registres bidirectionnel de m bascules chacun. Le nombre de niveaux m est appelé profondeur de la pile ; avec m niveau, il est possible de réaliser m appel successif à des procédures imbriqués.

Lorsque la profondeur m est la largeur de mots n deviennent importantes , on réalise en général la pile par l'assemblage d'une mémoire vive et d'un compteur, le pointeur de pile.

III.3 :Procédure avec paramètre :

Un ensemble de procédures de même forme, mais d'indices différents (variables de testes et/ou registres d'affectations) peut être remplacer par une procédure unique qui dépend de ces indices ; on l'appel : *procédure avec paramètre.*

Exemple 2 :

soit à détecter les valeurs des quatre bits X_0 , X_1 , X_2 , X_3 ; chaque valeur doit être mise dans un registre différent.

La figure (II.13-a) représente le DDB correspondant à cet exemple ; il se compose de quatre tranches qui ne se diffèrent entre elles que par les indices des variables d'entrées et des registres d'affectations. Ces quatre tranches peuvent êtres remplacées par une procédure unique de la figure (II.13-c) appelée à partir du programme principal de la figure (II.13-b).

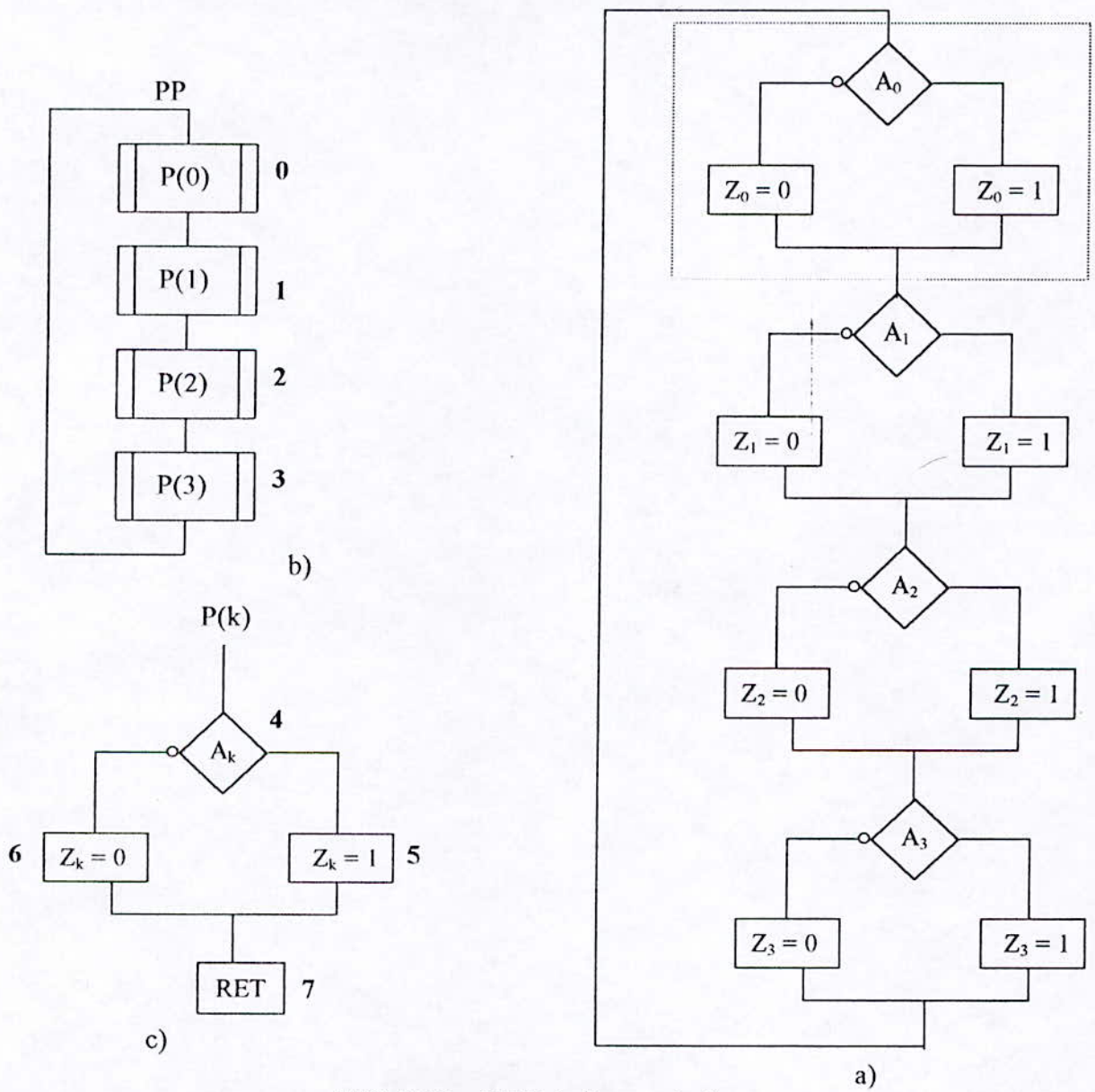


FIG.II.13 : DDB de l'exemple 2.

La définition du procédure avec paramètre implique une instruction d'appel avec paramètre , son expression mnémonique s'écrit sous la forme :

CALL ADSP (k) **RET TO** ADRT

On peut alors rédiger le microprogramme mnémonique comme suit :

ADR	Microprogramme mnémonique
0	CALL 4 (0) RET TO 1
1	CALL 4 (0) RET TO 1
2	CALL 4 (0) RET TO 1
3	CALL 4 (0) RET TO 1
4	IF A_k THEN 5 ELSE 6
5	DO REG(k) ← 0 GOTO 7
6	DO REG(k) ← 1 GOTO 7
7	RET

FIG.II.14 : microprogramme mnémonique de l'exemple 2.

Puisque à chaque valeur de l'indice k, pour cet exemple, correspond une seule variable d'entrée (variable de test), alors l'adressage de ces variable se fait seulement par la valeur de k, ainsi que les registres d'affectation.

Après le codage des instructions et les valeurs de l'indice k, le microprogramme en binaire s'écrit comme illustré dans la figure (II.15).

INST	CALL	IF	DO	RET
CODE (i_3 i_2)	00	01	10	11

K	0	1	2	3
CODE (i_1 i_0)	00	01	10	11

Pour exécuter ce programme il faudra disposer d'une pile auxiliaire semblable à la pile décrite ultérieurement avec le même nombre de niveaux (ici un seul niveaux suffit) empile ou dépile les valeurs de la variable k, le sommet k de cette pile auxiliaire commande le choix entre les variables d'entrée A_k , il commande également par un démultiplexeur, k registres de sortie distincts.

ADR	$i_3\ i_2\ i_1\ i_0$	ADR1 ADR+ ADSP	ADR0 OUT ADRT
0 0 0	0 0 0 0	1 0 0	0 0 1
0 0 1	0 0 0 1	1 0 0	0 1 0
0 1 0	0 0 1 0	1 0 0	0 1 1
0 1 1	0 0 1 1	1 0 0	0 0 0
1 0 0	0 1 - -	1 0 1	1 1 0
1 0 1	1 0 - -	1 1 1	- - 1
1 1 0	1 0 - -	1 1 1	- - 0
1 1 1	1 1 - -	- - -	- - -

ADR	ROM = [ADR]		
0	0	4	1
1	1	4	2
2	2	4	3
3	3	4	0
4	4	5	6
5	8	7	1
6	8	7	0
7	C	0	0

FIG.II.15 : microprogramme en binaire de l'exemple 2.

III.4 : Microprogramme avec compteur ordinal :

III.4.1- Définition:

Nous avons vu jusqu'ici que l'adressage des instructions du microprogramme était libre. Nous avons pu adresser les instructions là où elles se trouvaient, mais l'inconvénient c'est que le format des adresses était long et donc le nombre de bits d'un mot de la mémoire était trop grand . L'introduction d'un concept qui est le compteur ordinal ou séquenceur démunie pratiquement le nombre de bits des mots de la mémoire à la moitié . En effet, l'imposition des adresses incrémentées où chacune d'elles est déterminée à partir de la précédente , on ajoutant 1 accomplit la tâche. Dans l'instruction d'affectation il n'aura aucune adresse à spécifier puisque l'adresse futur sera l'adresse présente incrémentée de 1 et dans l'instruction de test, l'adresse futur du test vrai est l'adresse présente plus 1 . Ainsi, que pour l'instruction d'appel du sous microprogramme où l'adresse de retour au microprogramme principal ne sera plus spécifiée.

Les contraintes logicielles imposées par la spécification des adresses incrémentées seront compensées comme on l'a déjà dit, par de substantielles économiques matérielles qui sont :les restrictions des largeurs des instructions ; des problèmes beaucoup plus important que ceux ayant

des variables d'entrée et de nombre des états internes petits, pourront alors être traités par des interpréteurs de dimensions raisonnables.

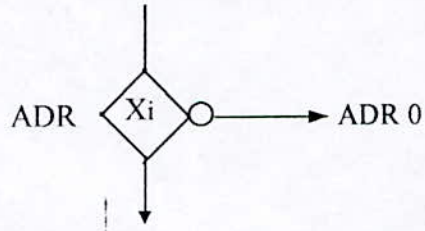
III.4.2- Micro-instructions mnémoniques pour un microprogramme incrémenté :

* micro-instruction de test :

IF Xi **ELSE** ADR0

SI Xi est vrai alors aller à ADR+1

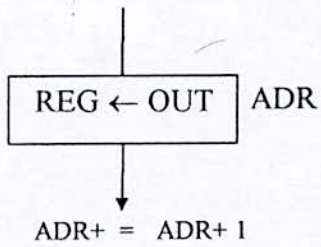
SI NON ADR0



* micro-instruction d'affectation :

D0 REG ← OUT

faire OUT dans REG et aller à ADR+1

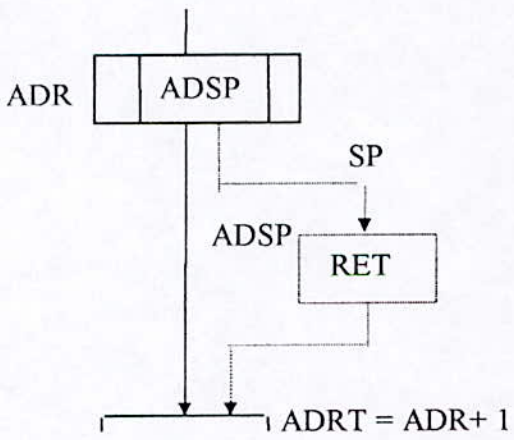


* micro-instruction d'appel :

CALL ADSP

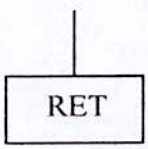
l'adresse de retour de l'instruction d'appel

vérifie la relation ADR = ADR+1



* micro-instruction de retour :

RET - inchangée



III.4.3- Séquenceur :

Un séquenceur à n bits est un système séquentiel synchrone destiné à calculer l'adresse futur ADR+ de n bits d'une mémoire de microprogramme, à partir de l'adresse présente (ADR), d'autres adresses (ADR, ADSP, ou STACK) et d'informations complémentaires constituées éssentiellement par les états de code opératoire.

Le séquenceur de la figure (II.16) est à huit bits, il est obtenu par la composition de deux circuits intégrés 74S482 de *TEXAS Instruments*. Ce séquenceur se décompose en une partie séquentielle comportant un registre d'adresses (ADR), une pile (STACK) et une partie combinatoire, formée par un multiplexeur (MUX) et un additionneur (ADD). Tous les bus transmettent des mots de huit bits :

* Le registre d'adresse (ADR) est défini par : OUT= ADR ← ADR+ = MUX;

* La pile (STACK) possède quatre opérations (HOLD, LOAD, POP, PUSH) qui sont commandés par les deux variables S3 et S4 selon la table des opérations suivantes :

opération	description : STACK	S3	S4
HOLD	NOP	0	0
LOAD	A ← ADD	0	1
POP	A← B, B←C, C←D, D←Q	1	0
PUSH	A← ADD, B←A, C←B, D ←C	1	1

* Le multiplexeur MUX est commandé par les variables S5 et S6, fait sélectionner l'adresse futur ADR+ à sa sortie parmi 4 mots possibles qui sont : l'entrée du séquenceur IN, la sortie de l'additionneur ADD, le sommet de la pile STACK et l'adresse présente ADR ; la table des opérations est la suivante :

opération	description	S5	S6
select IN	MUX = IN	0	0
select ADD	MUX = ADD	0	1
select STACK	MUX = STACK	1	0
select ADR	MUX = ADR	1	1

* L'additionneur (ADD) effectue l'addition arithmétique des trois mots : ADR (adresse présente), IN (entrée du séquenceur) et Cin (report éventuel) . Le résultat final se présente sous la forme de la somme calculée ADD et d'un éventuel report Cout. La table des opérations de cette additionneur est la suivante :

opération	description	S1	S2
ADD	Cout, ADD = ADR + IN + Cin	0	0
Incrément IN	Cout, ADD = IN + Cin	0	1
Incrément ADR	Cout, ADD = ADR + Cin	1	0
CARRY	Cout, ADD = Cin	1	1

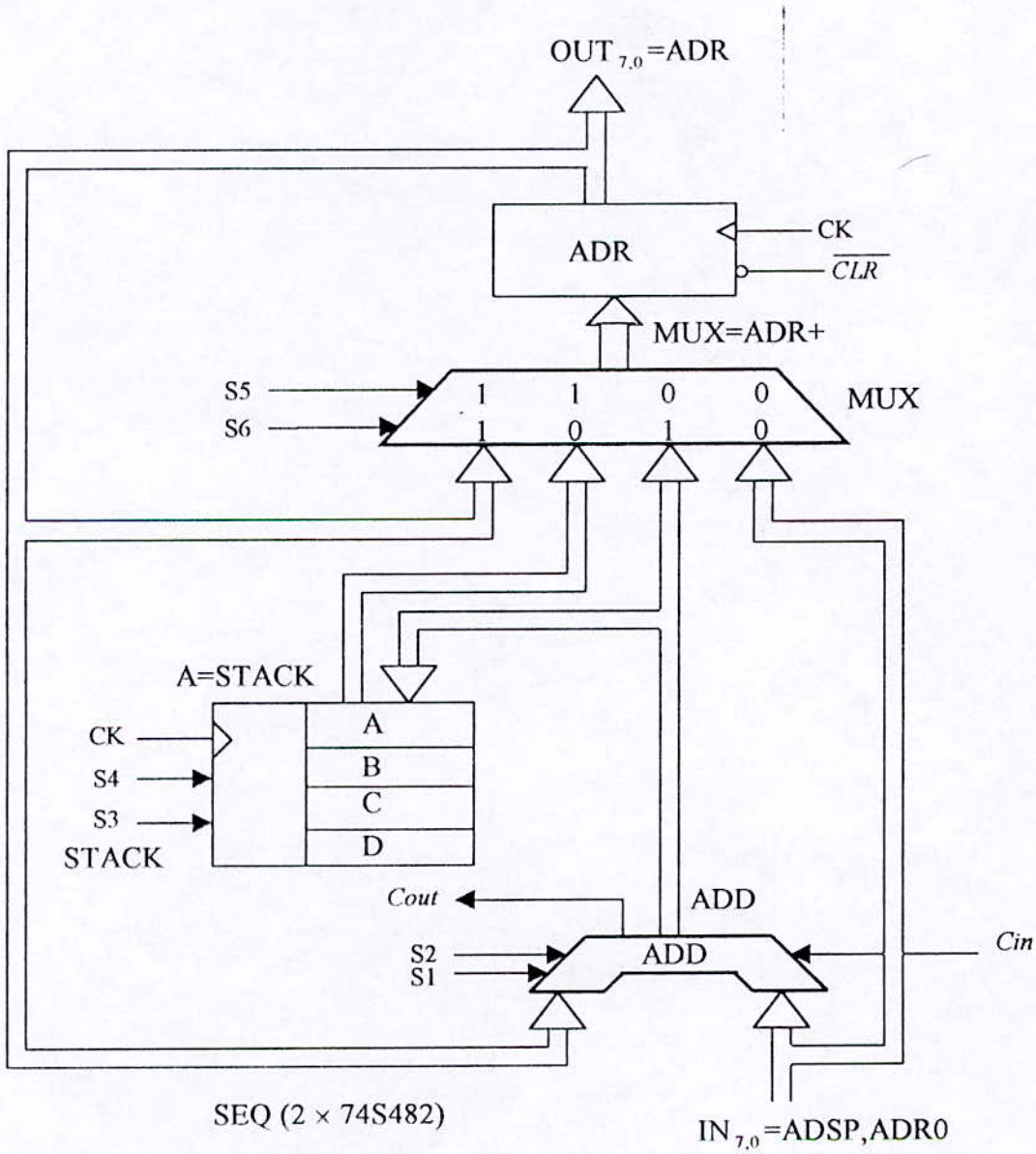


FIG.II.16 : séquenceur (2 × 74S482) de Texas Instrument.

les bus sont tous à 8 bits.

III.4.4- Instruction de saut inconditionnel :

On tombe des fois dans des cas où l'instruction qu'on veut adresser n'est pas celle incrémentée mais une autre se trouvant dans un endroit supérieur ou inférieur du microprogramme .La disposition des adresses des instructions du microprogramme impose cet inconvénient. L'introduction de la notion du saut inconditionnel fait le remède à cet inconvénient.

*** expression mnémonique :**

GOTO ADR0 (aller à l'adresse 0).

D'autre part un problème se pose : l'interprétation de cette instruction par les interpréteurs, deux solutions se présentent : la première c'est de coder cette nouvelle instruction, ce qui nous oblige à faire des modifications sur le matériel qui est un peut délicat ; la deuxième, c'est de voir que l'instruction précédente est équivalente à cette expression :

IF (0 = 1) **ELSE** ADR0

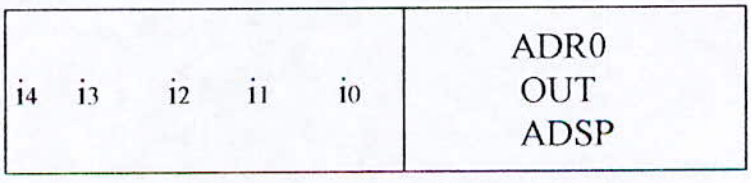
L'exploitation de cette équivalence nous oblige à introduire une variable d'entrée supplémentaire toujours forcée à zéro.

Dans l'exemple précédent (comparateur de bits) on introduit une deuxième variable y toujours à zéro. Le code opératoire résultant sera à 5 bits.

* Codage des variables d'entrée :

variables	Xo	X1	y
codes	0 0	0 1	1 0

Revenons à notre exemple du comparateur, le format résultant de la notion de microprogramme incrémenté sera le suivant :



i4 i3 : code instruction.

i2 i1 : codage des variables d'entrée.

io : codage du Test par rapport à 0 ou à 1.

Le microprogramme correspondant sera :

ADR	microprogramme mnémonique
0	**DO** REG ← 0
1	**CALL** 7
2	**CALL** 7
3	**CALL** 7
4	**CALL** 7
5	**DO** REG ← 1
6	**IF** y = 1 **ELSE** 0 (≡ **GOTO** 0)
7	**IF Xo = 1 ELSE A**
8	**IF X1 = 1 ELSE** 7
9	**IF** y = 1 **ELSE** B (≡ **GOTO** B)
A	**IF X1 = 0 ELSE** 7
B	**RET**

FIG. II.17 : microprogramme mnémonique.

Le microprogramme en binaire correspondant est le suivant :

ADR	i4 i3 i2 i1 i0	ADR 0 OUT ADSP
0 0 0 0	0 0 - - -	- - - 0
0 0 0 1	1 0 - - -	0 1 1 1
0 0 1 0	1 0 - - -	0 1 1 1
0 0 1 1	1 0 - - -	0 1 1 1
0 1 0 0	1 0 - - -	0 1 1 1
0 1 0 1	0 0 - - -	- - - -
0 1 1 0	0 1 1 0 1	0 0 0 0
0 1 1 1	0 1 0 0 1	1 0 1 0
1 0 0 0	0 1 0 1 1	0 1 1 1
1 0 0 1	0 1 1 0 1	1 0 1 1
1 0 1 0	0 1 0 1 0	0 1 1 1
1 0 1 1	1 1 - - -	- - - -

a)- microprogramme en binaire.

ADR	OPC	ADR 0 OUT ADSP
0	0 0	0
1	1 0	7
2	1 0	7
3	1 0	7
4	1 0	7
5	0 0	1
6	0 D	0
7	0 9	A
8	0 B	7
9	0 D	B
A	0 A	7
B	1 8	0

b)- en hexadécimal

FIG.II.18

IV. CONCLUSION:

Nous avons vu dans ce chapitre les concepts de microprogramme linéaire et non linéaire . Un microprogramme linéaire ayant des instructions répétitives consomme plus de place mémoire . L'élimination de ces opérations redondantes nous a obligé à introduire la notion du sous microprogramme . Ainsi, le microprogramme final se réduit à un microprogramme principal et des sous microprogrammes non répétitifs . Or l'appel à ces sous microprogrammes et l'assurance de revenir au microprogramme principal à l'adresse voulue nécessitait une solution matérielle : La pile .

La réalisation matérielle d'une pile associé aux nouvelles micro-instructions d'appel et de retour a fait plus de clarté sur les microprogrammes rédigés et surtout les possibilités de décomposition de problèmes plus complexes par des appels successifs de procédures (procédures imbriquées) , constituent les principaux avantages d'une telle solution.

Par ailleurs, la largeur des formats des micro-instructions introduites dans la mémoire était considérable, en effet ce format nécessitait la spécification de deux adresses pour la micro-instruction de test, l'adresse futur pour celle d'affectation et de deux adresse pour la micro-instruction d'appel.

Imposer certaines adresses (ADR1, ADR+, ADRT) à avoir des valeurs de l'adresse présente incrémenté de un donne lieu à un nouveau concept de microprogramme : microprogramme incrémenté.

L'introduction de cette contrainte logicielle, l'adressage incrémenté, nous a permis d'obtenir une réduction importante de la largeur des micro-instructions. Ce gain matériel se paie par une correction logicielle des microprogrammes, qui peut augmenter la profondeur de la mémoire finale.

La combinaison de ce nouveau logiciel, le microprogramme incrémenté et d'un nouvel opérateur matériel, le séquenceur , nous permet de réaliser un interpréteur caractérisé par une architecture très simple (l'assemblage d'une mémoire, d'un séquenceur et de quelques opérateurs logiques élémentaires) et capable d'exécuter des microprogrammes relativement complexes.

CHAPITRE III :

Conception d'un automate de commande d'une porte de sécurité à code

CHAPITRE III : CONCEPTION D'UN AUTOMATE DE COMMANDE D'UNE PORTE DE SECURITE A CODE :

I. INTRODUCTION :

Nous avons vu dans les chapitres précédents les différentes méthodes de description des machines séquentielles. Le graphe des états et la table ,des transitions sont très utilisés pour un nombre de variables et d'états internes réduits. La synthèse d'un système peut alors se limiter à cette étape en adoptant un modèle simple dit : modèle comportemental. Figure (III.1)

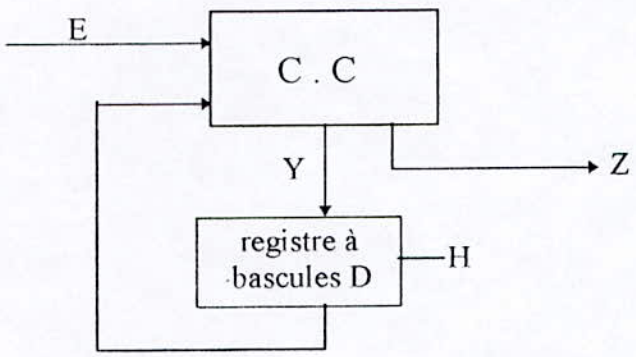


fig. III.1 : modele comportemental

Le bloc C.C est un circuit combinatoire qui peut être un décodeur (modèle câblé) ou une EPROM (modèle programmé). E représente l'entrée du système, Y l'état interne et Z la sortie.

Pour des automates plus complexes, le graphe des états est très difficile à mettre en œuvre à cause du nombre très considérable des états internes et, encore, la spécification des interactions entre différents variables n'est pas facile à mettre en œuvre.

On va voire dans ce chapitre pour notre synthèse de la commande d'une porte de sécurité à code que le système (à partir des cahiers des charges) présente plusieurs variables d'entrées et états internes, ce qui rend la construction du graphe très délicat; par exemple pour concevoir un système ayant cinq variables d'entrée et quatre variables d'états internes, on aura $2^4 \times 2^5 = 2^9 = 512$ combinaisons différentes, ce qui nécessitera une EPROM de $4 \times 512 = 2048$ bits plus huit décodeurs d'adresses (6×64).

Le modèle microprogrammé vient comme remède au défauts du modèle comportemental. En effet, la facilité de mettre en œuvre le DDB , même par des gens non spécialistes, et la souplesse logicielle-matérielle , le rend très puissant et très répondant aux automates de complexité considérable.

Un automate est généralement conçu pour commander une machine ou un groupe de machines. On appelle cette (ou ces) machines la « partie opérative » du processus, alors que l'ensemble des composants de l'automate fournissants les informations qui servent à piloter cette partie opérative qui est appelée « partie commande », on l'appelle aussi « automate ».

La machine reçoit des informations de l'automate, qui doit exécuter (par exemple : signaux d'ouverture et de fermeture d'une porte) et renvoi des informations à l'automate en fonction des résultats de ces actions (par ex : signal de fin de course), ces informations constituent les sorties de la machine qui sont les entrées de l'automate, la deuxième partie des entrées de l'automate étant l'ensemble des instructions transmises par l'opérateur (figure III.2).

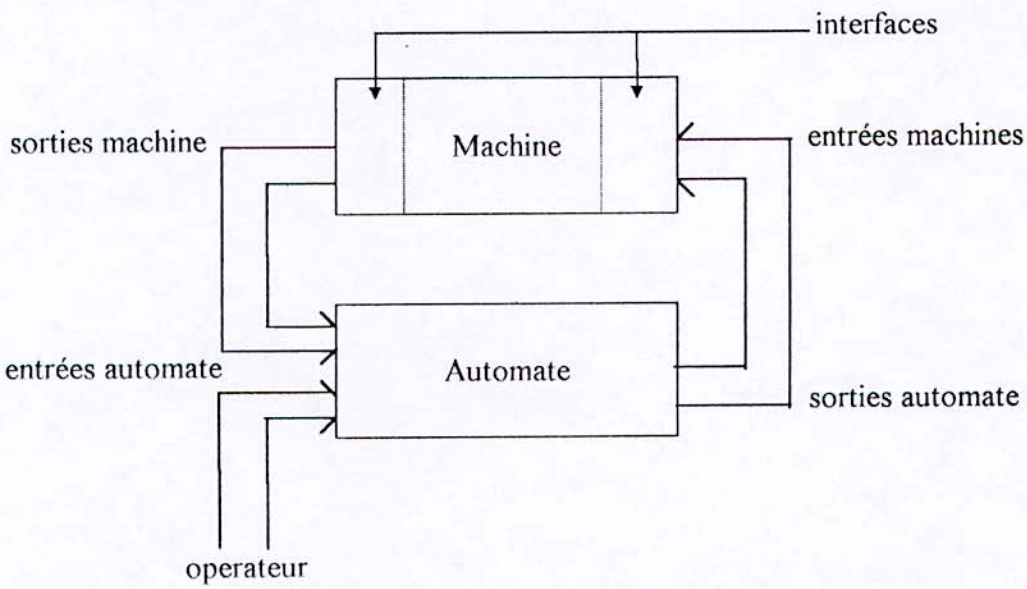


fig. II.2: schema général d'un automatisme

Outre la partie opérative et la partie commande, cette dernière échange des informations avec l'opérateur dont elle reçoit et à qu'il fournit des comptes rendus (sonores ou lumineux).

La conception d'un automate de commande d'une porte de sécurité à code est un exemple de synthèse des petits automates par microprogramme suivant un cahier des charges bien établi.

II. Cahier des charges :

Soit à concevoir un automate de commande de la porte qui assure le comportement général suivant:

-Affichage d'un message invitant à introduire le code secret.

- Programmation du code secret par roue codeuses.

- Introduction du code par clavier.

- Remise à zéro du système d'une façon automatique.

- Validation du code d'entrée manuellement.

- L'ouverture de la porte suite au bon code entré, et affichage d'un message de succès.

- Suite à un faux code entré, l'affichage d'un message invitant à introduire un nouveau code avant 16 secondes après lesquelles il y aura enclenchement d'une alarme de 8 secondes si le bon code n'a pas été introduit.

- La fermeture automatique de la porte après la fin de l'ouverture.

- Un maintient de l'ouverture à 16 sec (après de l'ouverture) si l'automate détecte un obstacle, et après lesquelles (16 sec) il y aura un déclenchement d'une alarme jusqu'à la disparition de l'obstacle.

- Fermeture automatique après disparition de l'obstacle détecté.

Remarque : Pour le fonctionnement correct de l'automate, on doit disposer de certains capteurs qui sont :

- Un capteur qui délivre un signal de fin de course pour indiquer la fermeture totale de la porte.

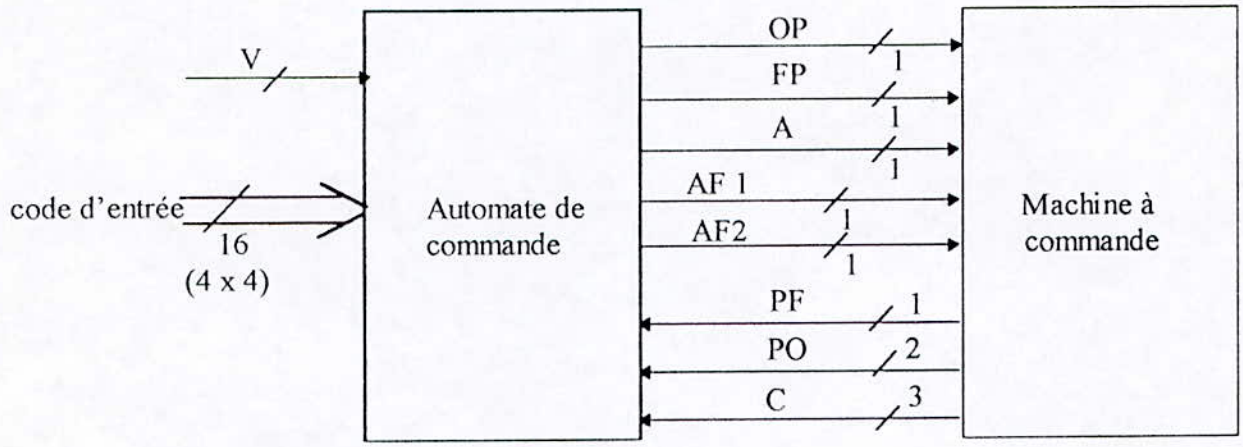
-Un capteur qui délivre un signal de fin de course pour indiquer l'ouverture totale de la porte.

- Un capteur qui détecte la présence d'un obstacle.

III - Schéma fonctionnel :

Comme tout automatisme, notre système sera constitué de deux parties essentielles : la partie commande (ou l'automate de commande) et la partie opérative (ou la machine à

commande) entre les quelles et l'opérateur(l'utilisateur), vont être véhiculées des informations ou des signaux qui vont gérer le fonctionnement de l'automatisme, comme l'illustre la figure suivante



V : signal de Validation du code entré .

OP : signal commandant l'ouverture .

FP : signal commandant la fermeture .

A : signal d'activation de la larme .

PF : signal indiquant la fermeture complète .

PO : signal indiquant l'ouverture totale .

OB : signal indiquant la présence d'un obstacle .

AF1 : signal commandant l'affichage de «entrer le code ».(lampe verte).

AF2 : signal commandant l'affichage de « refaire le code ».(lampe rouge).

En plus de ces signaux, le fonctionnement du système est régit par d'autres signaux qui sont :

T : signal d'activation des temporisations .

T1 : signal indiquant la fin de la temporisation de 16 secondes .

T2 : signal indiquant la fin d'une temporisation de 8 secondes .

D : signal indiquant à l'automate que le premier chiffre a été introduit .

CV : signal indiquant à l'automate que le code introduit est valide .

Tous ces signaux peuvent être regroupés en deux types de signaux :

* les signaux d'état : ce sont des signaux que l'automate doit tester à chaque fois et suivant les quels l'état future ou le comportement du système sera fonction. Ces signaux sont :V, PF, PO, OB, T1, T2, D et CV.

* Les signaux de commande : Ce sont des signaux provenant de l'automate de commande et qui servent à la commande de la machine et de ces périphériques. Ces signaux sont : OP, FP, A, AF1, AF2.

IV. Schéma synoptique :

Suivant le comportement ou le fonctionnement de notre système décrit antérieurement, nous aurons le schéma synoptique de la figure (III.3).

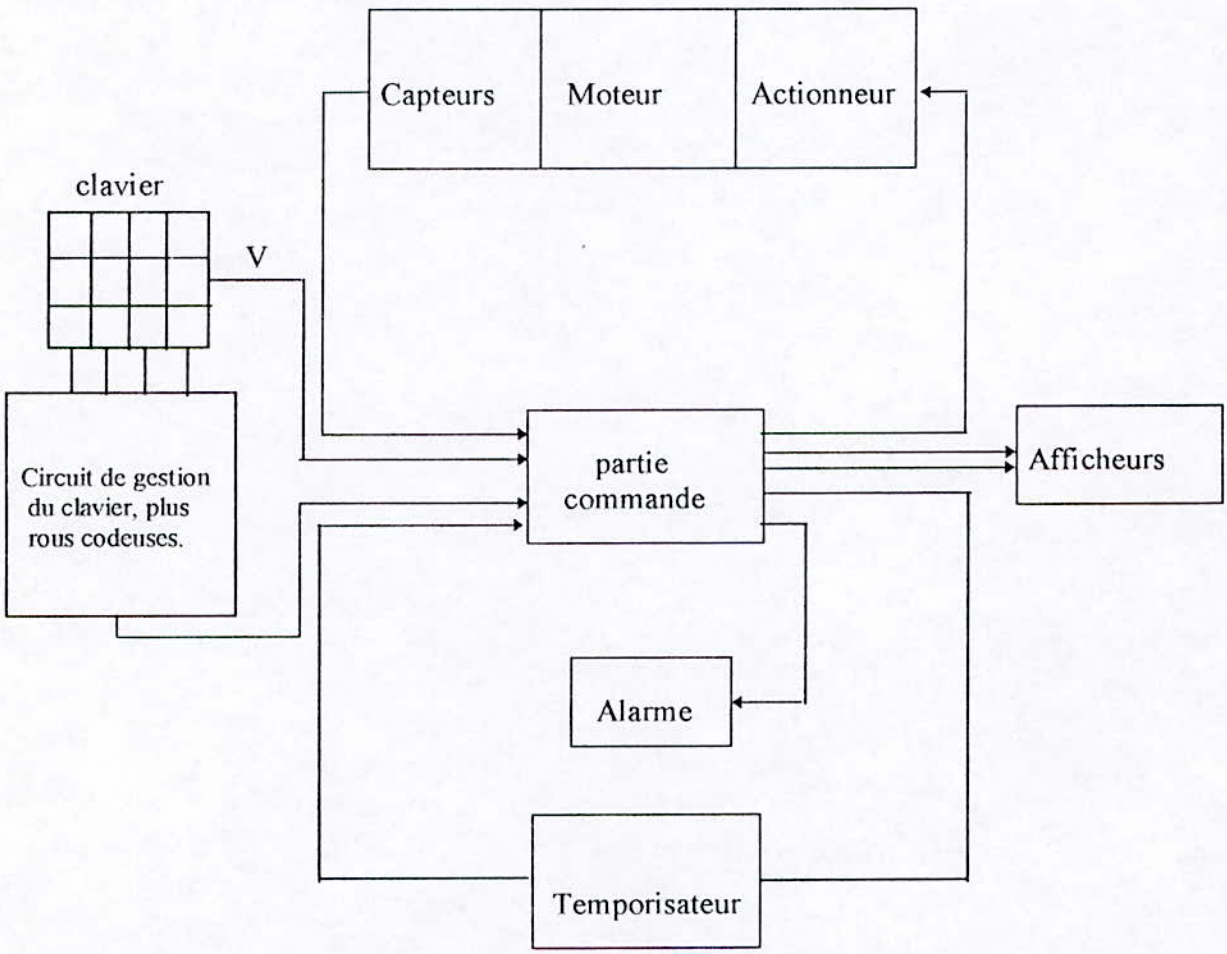


FIG.III.3 : schéma synoptique du système à concevoir.

V- Graphe de décision binaire et organigramme :

V-1.Organigamme :

Avant de faire la conception de n'importe quel automatisme, il faut avant toutes choses passer par le graphe de décision binaire, pour spécifier les différents états internes ainsi que les variables régissent le fonctionnement de l'automatisme, et d'où la facilité de prévoir et de bien définir ses exigences matérielles et logicielles ; qui convertissent pour les 1^{eres} par l'architecture électrique et les différents composants constitutifs avec leurs interconnexions, et pour les 2^{emes} par le codage et l'adressage des différentes instructions constituants le microprogramme mnémonique qui gère le fonctionnement de cet automatisme.

Mais puisque le graphe de décision binaire n'est autre qu'une forme logique d'un organigramme qui n'est à son tour qu'une forme organisée du cahier des charges, alors un graphe de décision binaire simplifié et bien organisé ne sera que le résultat d'un bon organigramme.

La figure (III.4) représente l'organigramme décrivant le fonctionnement de notre automatisme suivant les exigences du cahier des charges.

Remarque : La partie (*) de l'organigramme correspond à la remise à zéro automatique.

V.2 : Graphe de décision binaire :

On plus, des signaux de commande que nous avons déjà défini, on défini un autre signal RZ qui sert comme signal de remis à zéro des différents composants du système.

Donc nous aurons un mot de commande constitué de 7 bits, disposés comme suit:

1^{er} bit	2^{eme} bit	3^{eme} bit	4^{eme} bit	5^{eme} bit	6^{eme} bit	7^{eme} bit ≡ mot de commande
CLR	OP	FP	A	T	AF1	AF2

Donc nous aurons le graphe de décision binaire de la figure.III.5 :

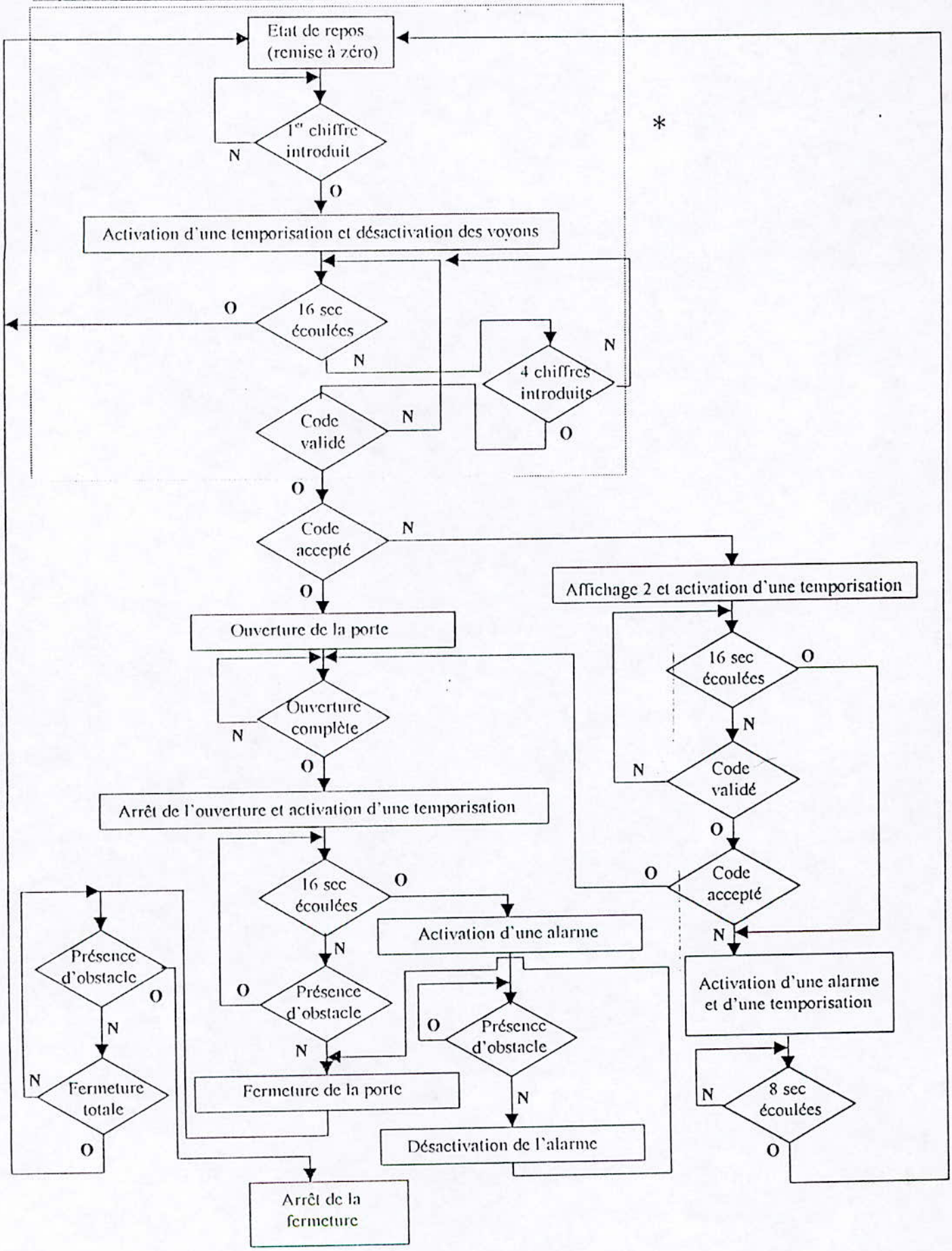


FIG.III.4 : organigramme de l'automate.

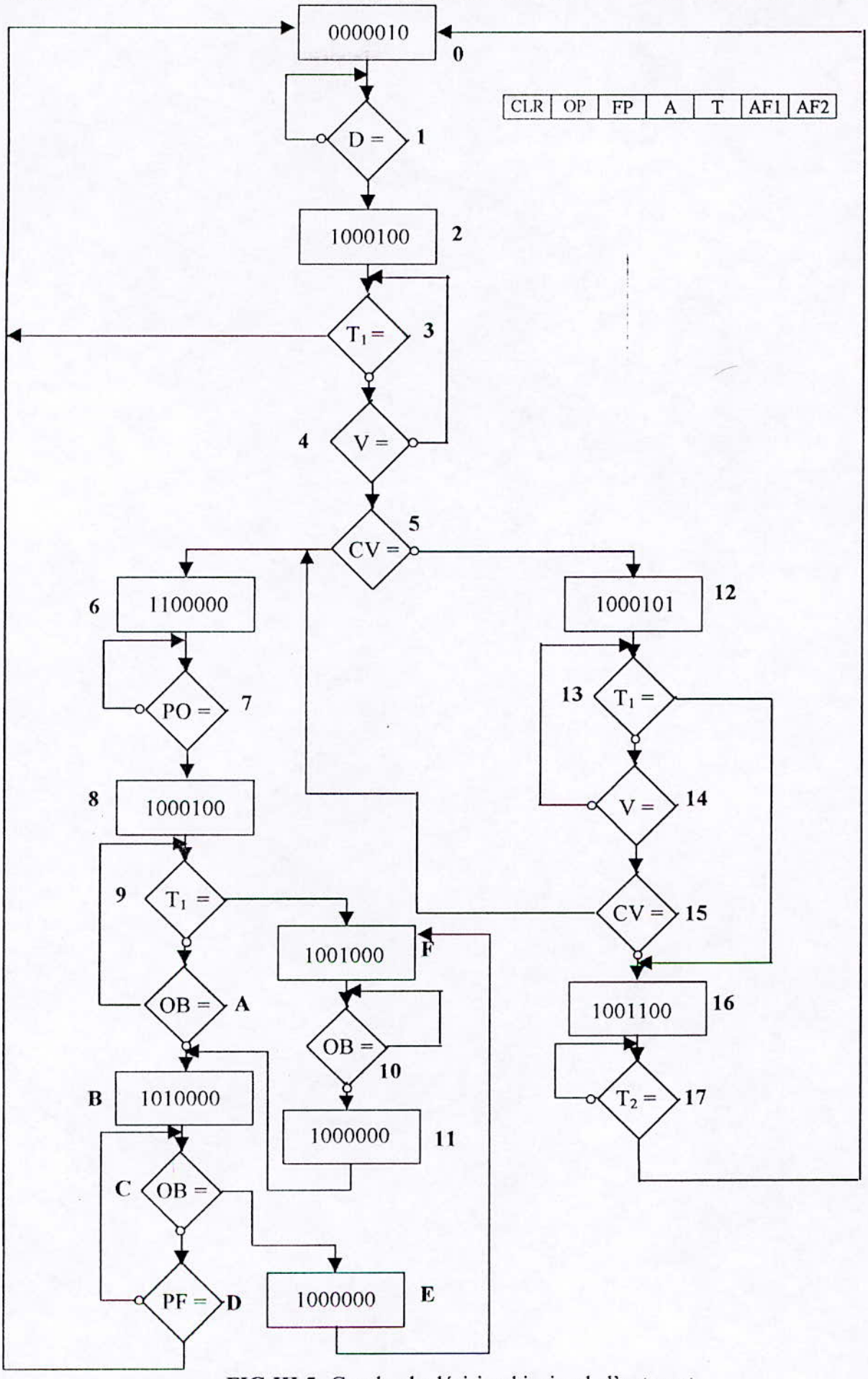


FIG.III.5 :Graphe de décision binaire de l'automate.

VI- Microprogramme mnémonique :

Le programme mnémonique de notre système sera comme suit:

ADRESSE	microprogramme mnémonique					
0	**DO**	RG ← 0000010	**GOTO**	1		
1	**IF**	D	**THEN**	2	**ELSE**	1
2	**DO**	RG ←1000100	**GOTO**	3		
3	**IF**	T1	**THEN**	0	**ELSE**	4
4	**IF**	V	**THEN**	5	**ELSE**	3
5	**IF**	CV	**THEN**	6	**ELSE**	12
6	**DO**	RG ← 1100000	**GOTO**	7		
7	**IF**	PO	**THEN**	8	**ELSE**	7
8	**DO**	RG ← 1000100	**GOTO**	9		
9	**IF**	T1	**THEN**	F	**ELSE**	A
A	**IF**	OB	**THEN**	9	**ELSE**	B
B	**DO**	RG ← 1010000	**GOTO**	C		
C	**IF**	OB	**THEN**	E	**ELSE**	D
D	**IF**	PF	**THEN**	0	**ELSE**	C
E	**DO**	RG ← 1000000	**GOTO**	F		
F	**DO**	RG ← 1001000	**GOTO**	10		
10	**IF**	OB	**THEN**	10	**ELSE**	11
11	**DO**	RG ← 1000000	**GOTO**	B		
12	**DO**	RG ← 1000101	**GOTO**	13		
13	**IF**	T1	**THEN**	16	**ELSE**	14
14	**IF**	V	**THEN**	15	**ELSE**	13
15	**IF**	CV	**THEN**	6	**ELSE**	16
16	**DO**	RG ← 1001100	**GOTO**	17		
17	**IF**	T2	**THEN**	0	**ELSE**	17

VII. Microprogramme en binaire et hexadécimal :

Pour rédiger le microprogramme binaire, on a coder les micro-instructions ainsi que les variables de test.

Codage des variables de test:

Variable	code
V	0 0 0
PF	0 0 1
PO	0 1 0
0B	0 1 1
CV	1 0 0
T1	1 0 1
T2	1 1 0
D	1 1 1

codage des instructions:

Instruction	code
IF	0
DO.....	1

VIII - Réalisation matériel : L'interpréteur :

L'ensemble des quatre instructions de test, d'affectation, constitue le langage mnémonique représenté symboliquement par : **L = { IF ... THENE ...ELSE , DO ... GOTO }**.

Le système séquentiel synchrone désigné à exécuter les deux micro-instructions de test et d'affectation est une machine de décision binaire, appelé aussi interpréteur . Il se compose de :

- Une mémoire de programme compatible avec le format des instructions imposées
- Un registre d 'adresse ADR dont l'état ADR est l'adresse de la mémoire ; il est défini par la relation ADR ← ADR+ .
- Un premier multiplexeur (MUX1), commandé par la variable y et destiné à sélectionner l'adresse futur ADR+ .
- Un second multiplexeur (MUX2), commandé par la variable du code de l'instruction et par les variables d'entrée (leurs codes) et destiné à sélectionner la variable de test Xi.
- Un registre d'affectation ou de sortie REG qui constitue un cas particulier du registre bidirectionnel.

microprogramme binaire				Hexadécimal			
Adresse		ADR 1 ADR + ADSP	ADR 0 OUT ADRT	ADR	ROM = [ADR]		
00000	1 - - -	00001	0000010	0	8	01	02
00001	0111	00010	- - 00001	1	7	02	01
00010	1 - - -	00011	1000100	2	8	03	44
00011	0101	00000	- - 00100	3	5	00	04
00100	0000	00101	- - 00011	4	0	05	03
00101	0100	00110	- - 10010	5	4	06	12
00110	1 - - -	00111	1100000	6	8	07	60
00111	0010	01000	--00111	7	2	08	07
01000	1 - - -	01001	1000100	8	8	09	44
01001	0101	01111	- - 01010	9	5	0F	0A
01010	0011	01001	- - 01011	A	3	09	0B
01011	1 - - -	01100	1010000	B	8	0C	50
01100	0011	01110	- - 01101	C	3	0E	0D
01101	0001	00000	- -01100	D	1	00	0C
01110	1 - - -	01111	1000000	E	8	0F	40
01111	1 - - -	10000	1001000	F	8	10	48
10000	0011	10000	- - 10001	10	3	10	11
10001	1 - - -	01011	1000000	11	8	0B	40
10010	1 - - -	10011	1000101	12	8	13	45
10011	0101	10110	- -10100	13	5	16	14
10100	0000	10101	- -10011	14	0	15	13
10101	0100	00110	- -10110	15	4	06	16
10110	1 - - -	10111	1001100	16	8	17	4C
10111	0110	00000	- - 10111	17	6	00	17

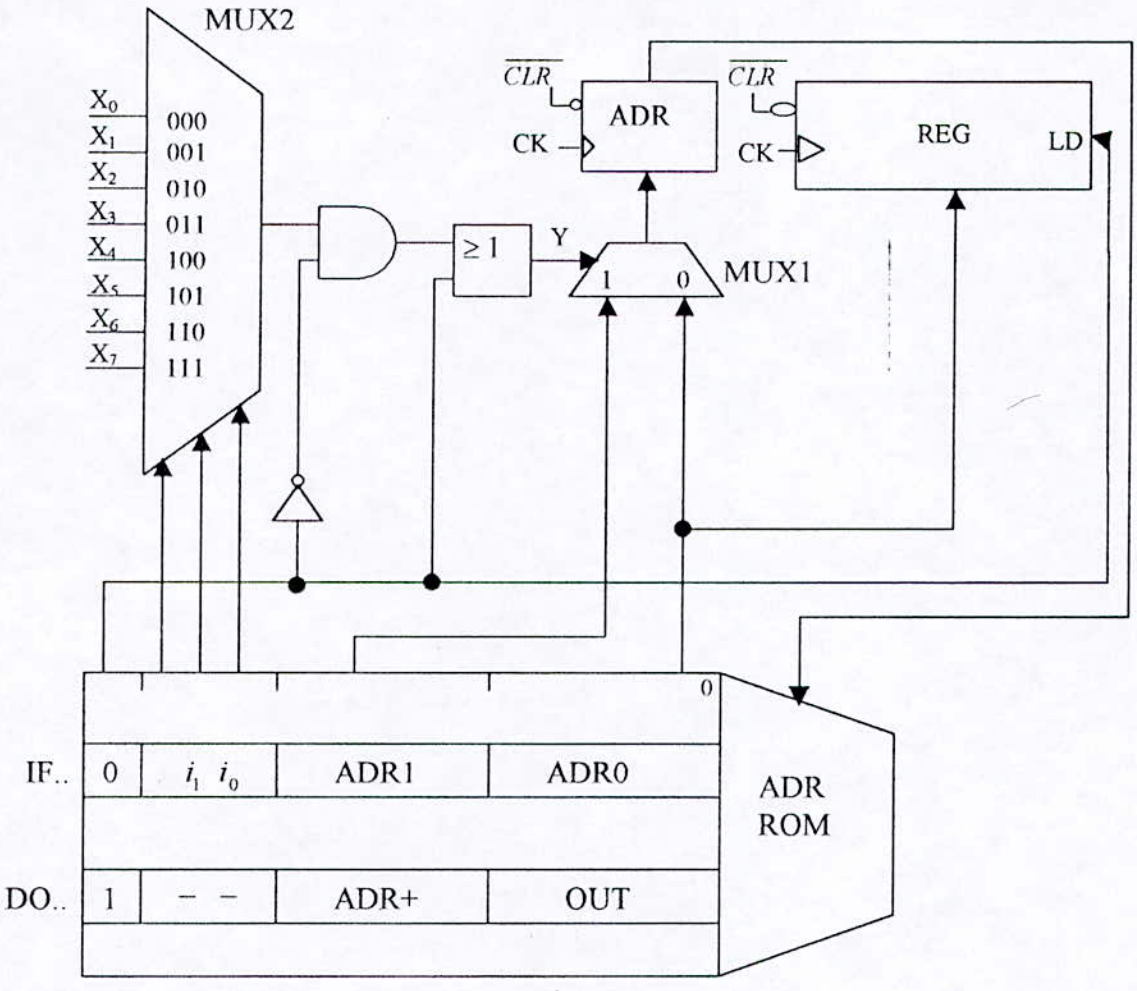


FIG.III.6 : interpréteur des deux micro-instructions de test et d'affectation.

IX. Schéma électrique :

Le schémas de notre système (figure : III.7) est relativement simple, en effet le cœur (ou le processeur) de notre automate est l'interpréteur **INTL** constitué de l'EPROM, des deux démultiplexeurs DMUX1 et DMUX2 et des deux registres : (d'adresse future) **ADR** et (d'affectation) **REG**, dont le fonctionnement a été décrit ultérieurement, est cadencé par une horloge assez rapide assurant le bon fonctionnement de notre système.

Dans la figure:(III.7) on n'a pas fait tout le détail de notre système à cause de l'encombrement, mais qui va être fait dans ce qui suit. Commençons –nous avant toutes choses de décrire le fonctionnement de la circuiterie de gestion du clavier, du fait de son importance puisqu'il est le seul moyen de l'opérateur pour commander l'automate.

Circuit de gestion du clavier :

Le circuit de gestion du clavier schématisé par la (figure :III.8) est très simple en structure et fonctionnement, en effet :

Le décodeur décimal - DCB fait le décodage des touches d'un clavier à touches indépendantes qui d'un coté sont reliées à la masse et de l'autre coté aux entrées inverseuses du décodeur.

L'ensemble des résistances liées aux entrées du décodeur assure le rappel à l'alimentation de chacune d'elles.

Un comparateur binaire (4 bits) fait la comparaison à chaque fois entre le chiffre introduit et le chiffre fourni par la roue codeuse correspondante .

A fin d'assurer la comparaison à chaque fois entre le bon chiffre introduit et le bon chiffre fourni par l'une des roues codeuses ; on a utilisé un registre à décalage de 4 bits qui à chaque état de repos ou reinitialisation de l'automate il reçoit à sa broche de chargement D_0 une impulsion venant du signal de remise à zéro, puis dés l'introduction du premier chiffre il reçoit une impulsion à sa broche d'horloge grâce au signal venant de la porte logique OR dont les entrées sont connectées aux sorties du décodeur et la touche correspondant au chiffre " zéro " après inversion par un inverseur, donc cette impulsion va faire évoluer le registre de l'état de sortie $Q_3Q_2Q_1Q_0$ =1000 à l'état $Q_3Q_2Q_1Q_0$ =0001 ce qui activera la roue codeuse 1 correspondant au premier chiffre du code secret et par conséquence la comparaison avec le premier chiffre introduit.

L'introduction du deuxième chiffre fera envoyer un nouveau signal d'horloge au registre qui se met à l'état $Q_3Q_2Q_1Q_0$ =0010 ce qui activera la deuxième roue codeuse et par conséquent la comparaison avec le deuxième chiffre introduit, et ainsi de suite jusqu'au quatrième chiffre.

Après comparaison, il y 'aura une impulsion qui sera envoyée à l'entrée CLK du compteur binaire modulo 4 à chaque introduction d'un bon chiffre. Notons ici qu'à fin de contrôler les sorties des roues codeuses, on a relié les sorties du registres de décalage aux broches d'alimentation de ces dernières.

Le signal CV indiquant à l'automate que le code introduit est valide sera fournit par la broche Q_2 du compteur binaire modulo 4 .

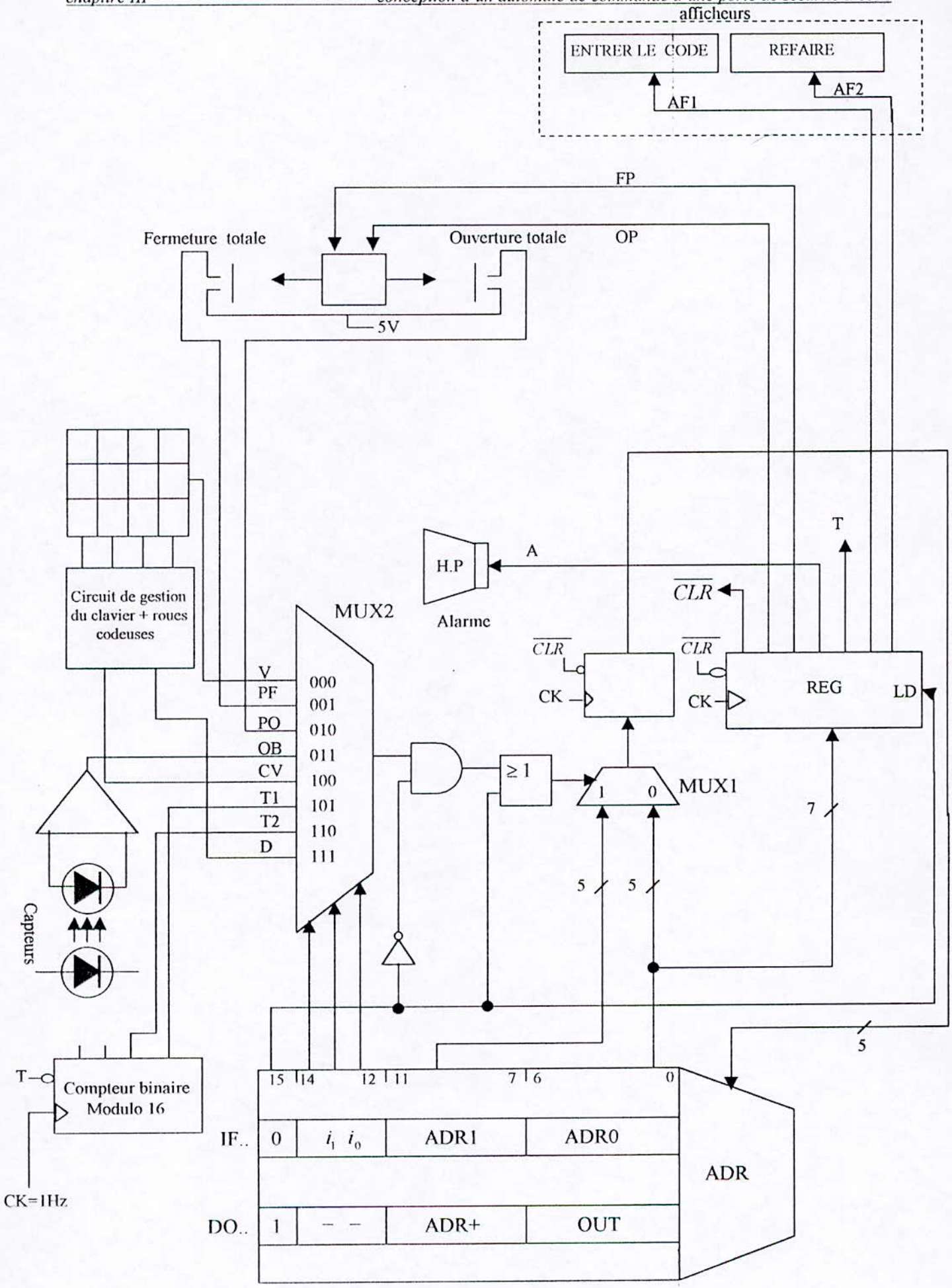


FIG.III.7 :schémas électrique

Notons aussi que le signal D venant de la porte OR permet à l'automate de détecter le premier chiffre introduit.

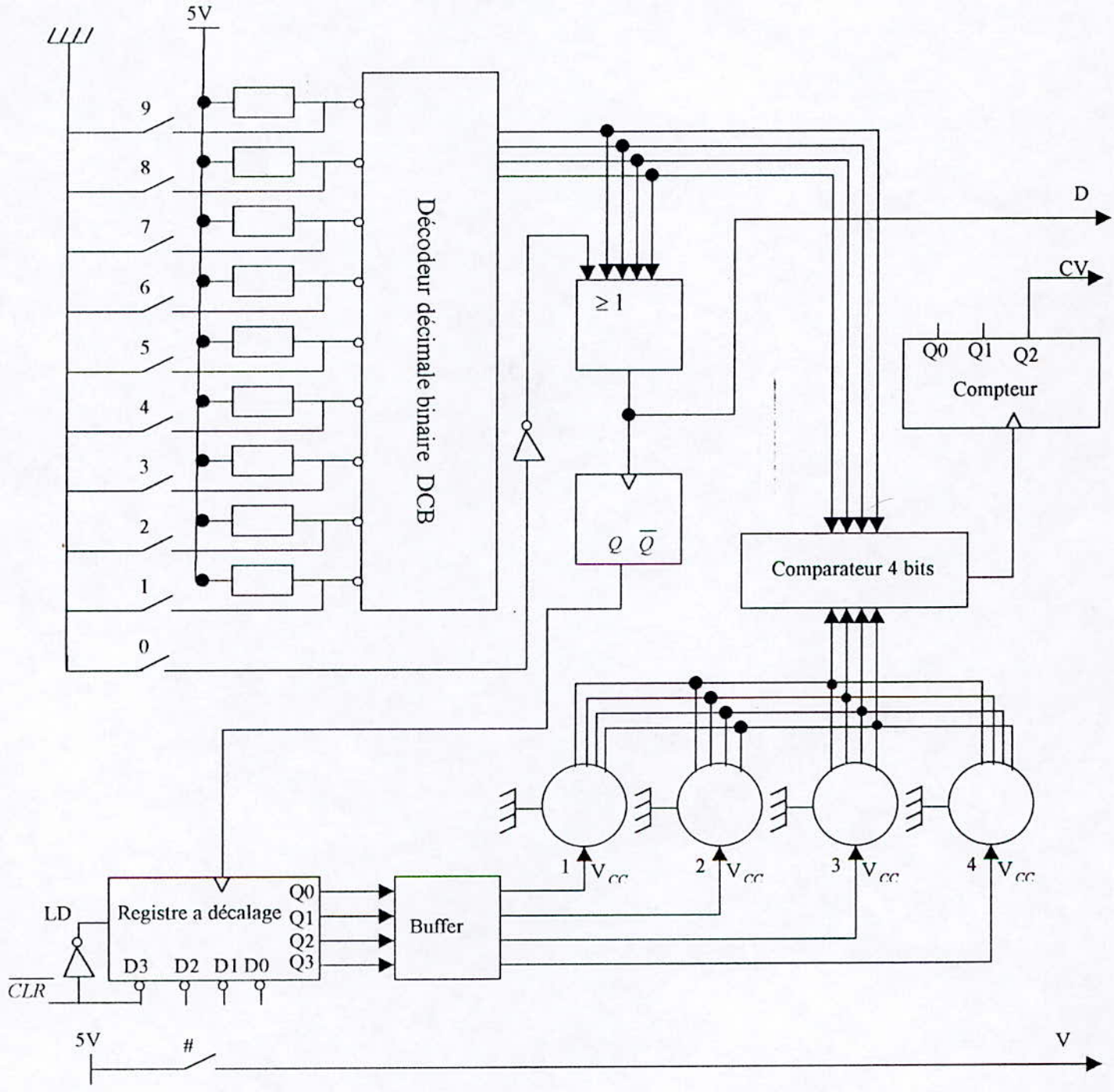


FIG.III.8 :circuit de gestion du clavier.

X. Comparaison par microprogramme :

Nous avons deux mots à comparer (A et B) de quatre chiffres, ces chiffres seront différenciés entre eux par un indice *l*, soient A_l , B_l , chaque chiffre se compose de quatre bits, pour les distingués on a besoin d'un deuxième indice *k*, soient A_{lk} , B_{lk} .

Le DDB de la figure (III.8) correspond à une tranche (comparaison du chiffre A_l avec B_l) parmi les quatre tranches du diagramme complet.

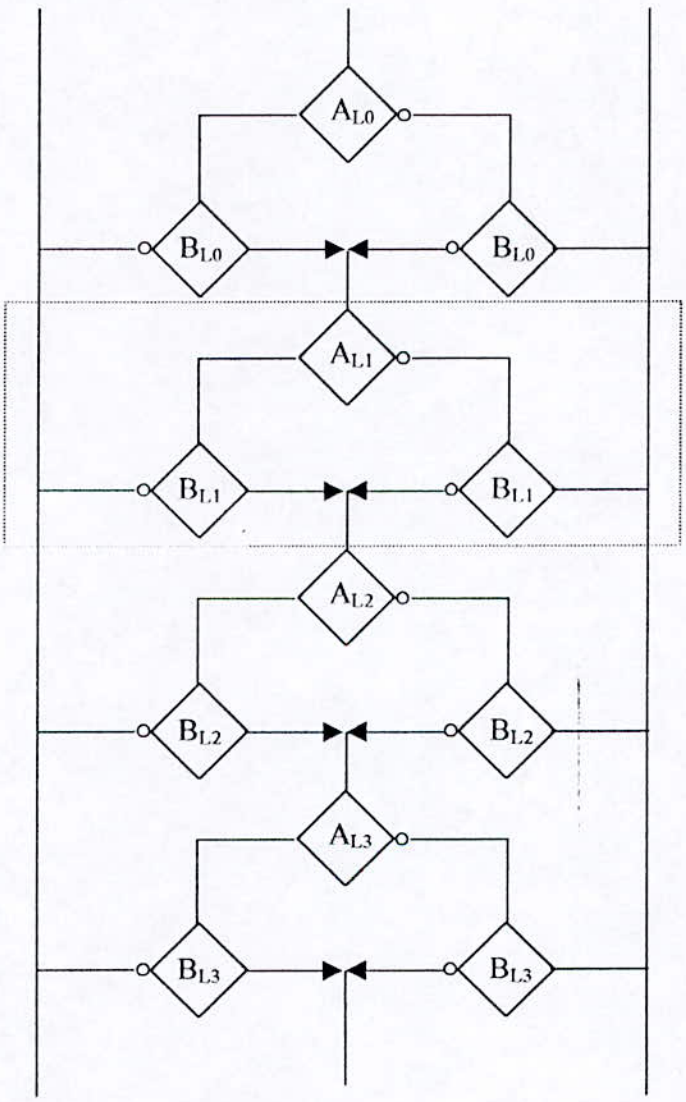


FIG.III.9 :comparaison du chiffre A_L avec le chiffre B_L.

Cette même tranche peut être diviser en quatre sous tranches (comparaison des bits A_{lk} avec B_{lk}), la tranche élémentaire est représentée en trait mixte dans la figure(III.9), elle à une borne d'entrée et trois bornes de sortie, or, pour exploiter la notion du procédure avec paramètre , cette tranche élémentaire doit avoir une borne de sortie unique. Pour cela on va affecter le

résultat de chaque comparaison de deux bits dans un registre intermédiaire. Le programme de comparaison des quatre bits d'un chiffre est un sous programme du programme principal. Le programme final se réduit donc à un programme principal et deux sous microprogramme comme illustré dans la figure (III.10).

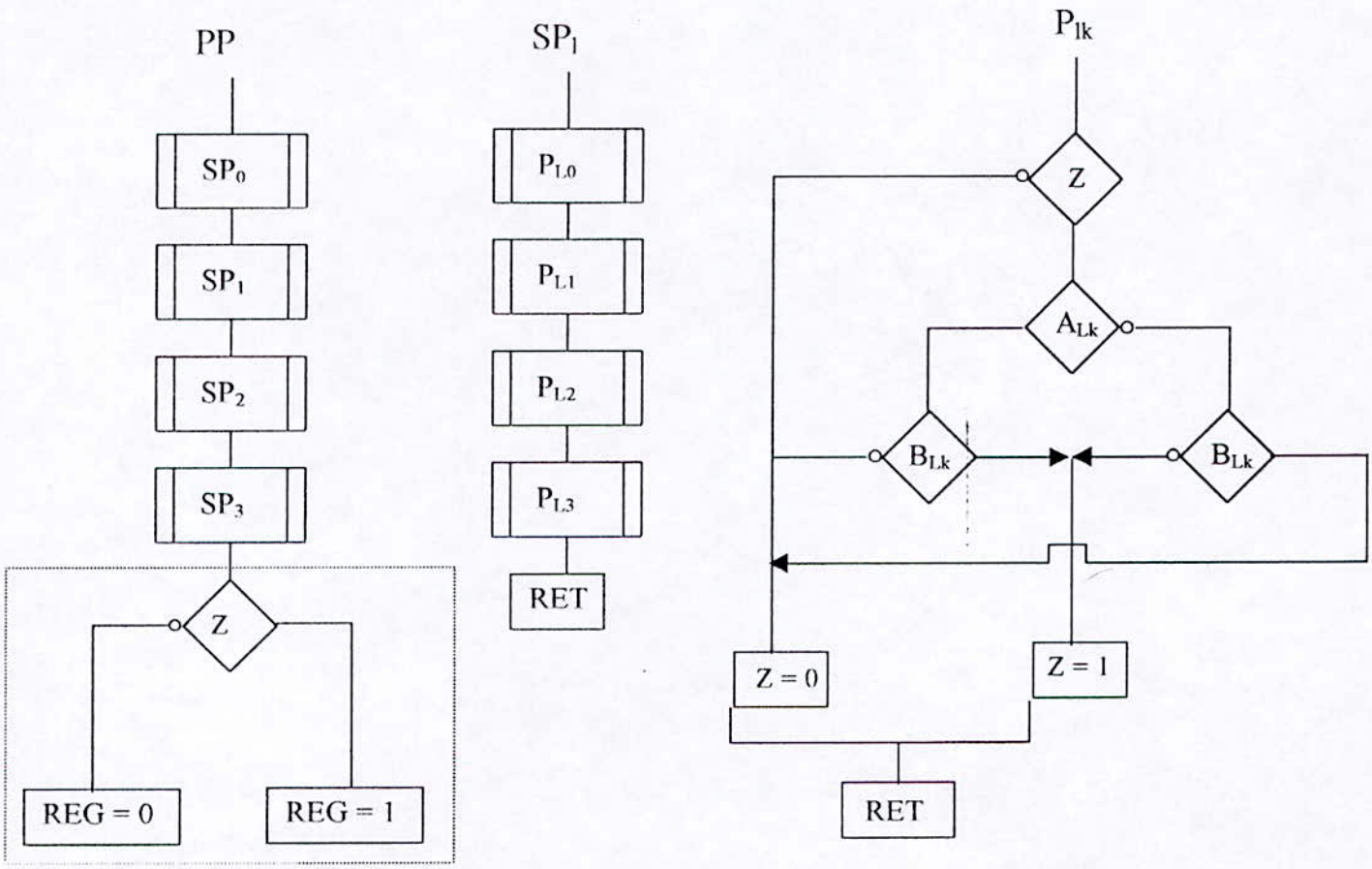


FIG.III.10 :programme de comparaison de 4 chiffres.

Où REG est le registre de sortie et Z un registre intermédiaire.

L'introduction de la comparaison par microprogramme dans le programme principal va nous faire économiser le comparateur de la figure (III.7), ainsi que les deux compteurs, mais, par contre, il exige un rajout du matériel dans la machine qu'on va le découvrir antérieurement ; le diagramme de décision binaire sera modifié comme il est représenté à la figure (III.11).

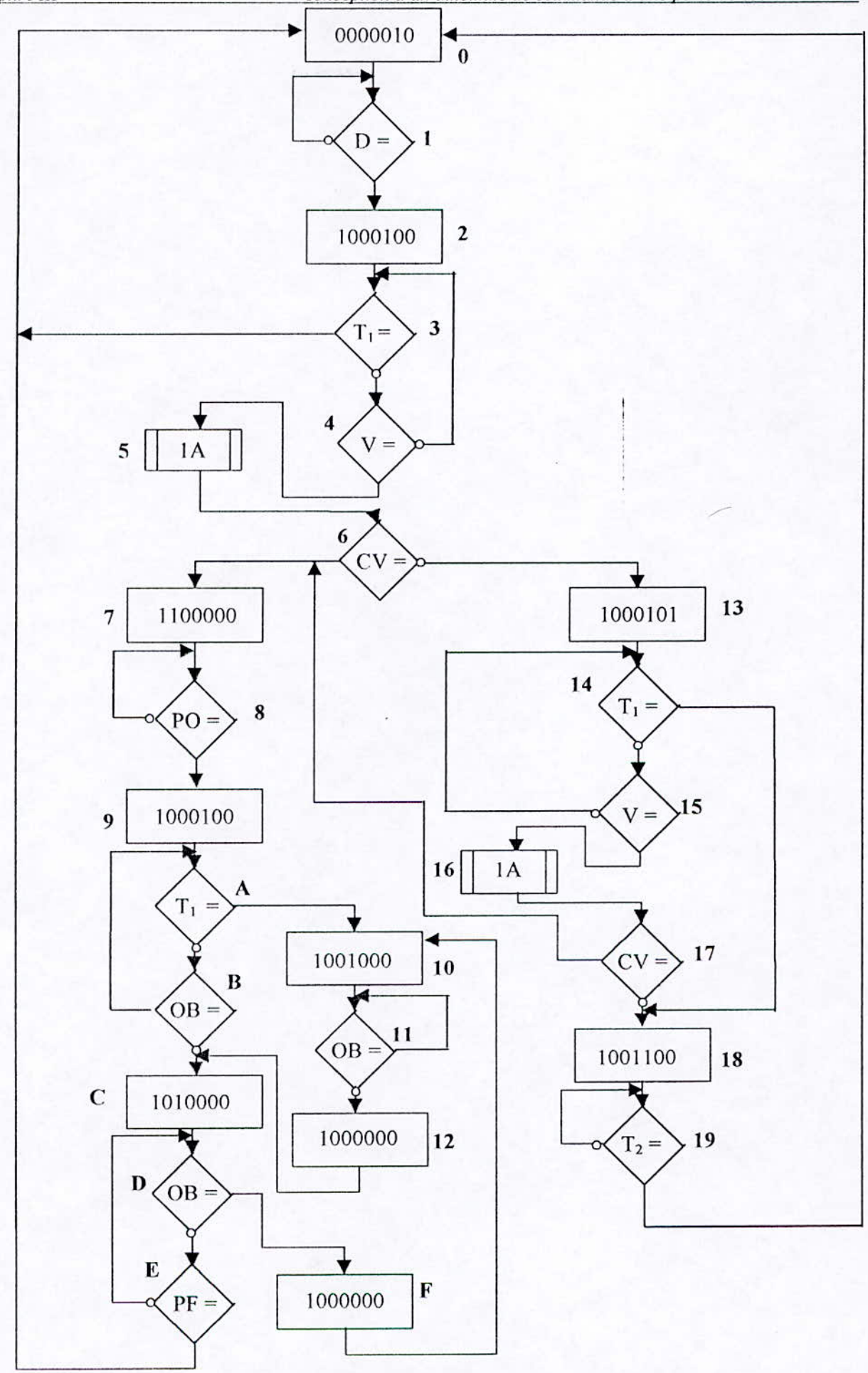
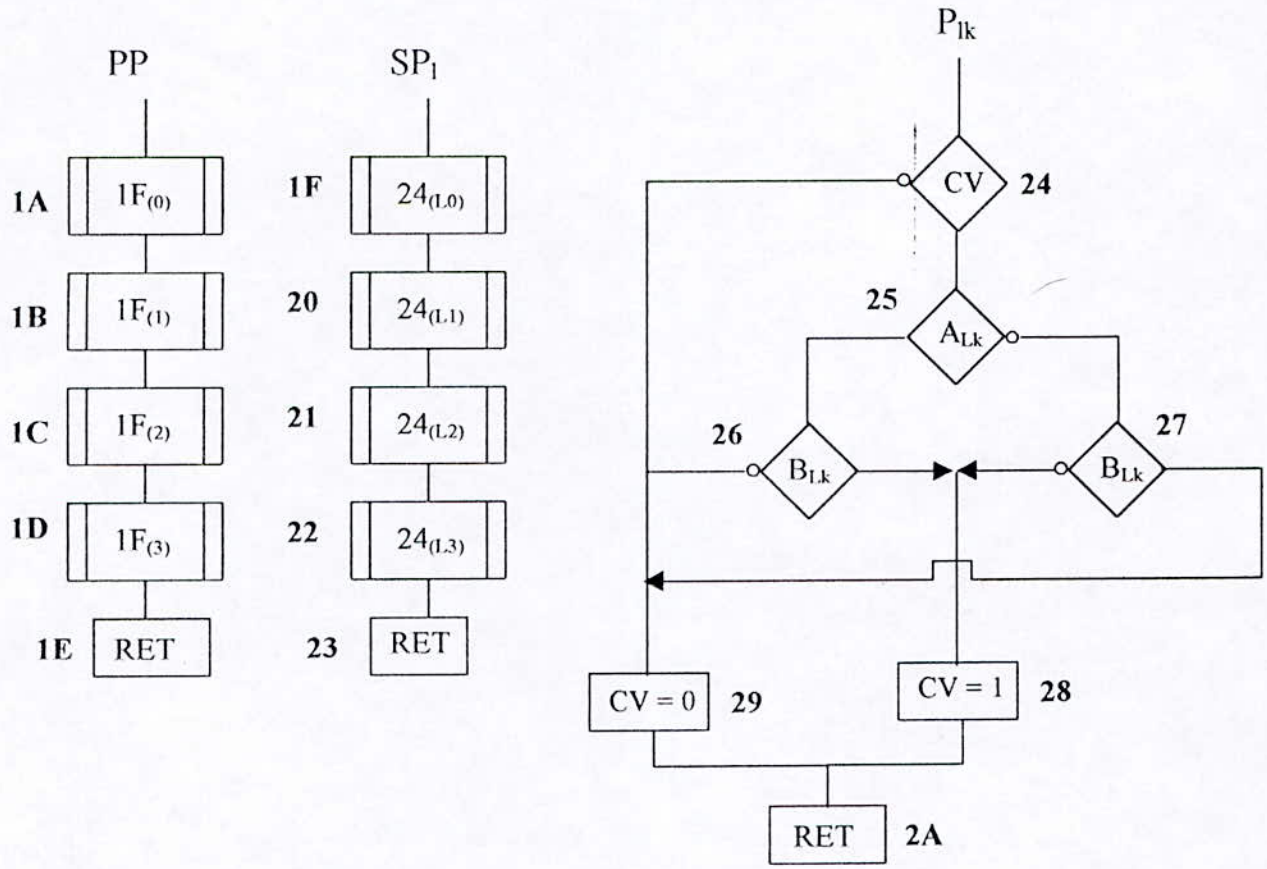


FIG.III.11 :Graphe de décision binaire de l'automate.



Graphe de décision binaire de l'automate (suite).

En utilisant les expressions mnémoniques des micro-instructions de test, d'affectation, d'appel de procédure (avec et sans paramètres) et de retour, nous rédigeons le microprogramme mnémonique représenté ci-après.

La rédaction du microprogramme en binaire nécessite une format bien établi pour que la machine soit capable de l'interpréter et de l'exécuter. Dans notre microprogramme, l'existence de différent type de procédures (procédure simple, avec paramètres, imbriquée) et des différent registres (registre de sortie et registres intermédiaires) ainsi que plusieurs variables d'entrée (variables de test du système et variables de test de comparaison des deux mots A et B), nous oblige à faire la différence entre eux. Pour cela nous proposons le format découlant des codages suivants :

❖ Codage des micro-instructions :

INST	CALL	IF	DO	RET
CODE	00	01	10	11

ADRESSE	microprogramme mnémonique					
0	**DO**	RG ← 0000010	**GOTO**	1		
1	**IF**	D	**THEN**	2	**ELSE**	1
2	**DO**	RG ←1000100	**GOTO**	3		
3	**IF**	T1	**THEN**	0	**ELSE**	4
4	**IF**	V	**THEN**	5	**ELSE**	3
5	**CALL**		1A	**RET TO**		6
6	**IF**	CV	**THEN**	7	**ELSE**	13
7	**DO**	RG ← 1100000	**GOTO**	8		
8	**IF**	PO	**THEN**	9	**ELSE**	8
9	**DO**	RG ← 1000100	**GOTO**	A		
A	**IF**	T1	**THEN**	10	**ELSE**	B
B	**IF**	OB	**THEN**	10	**ELSE**	C
C	**DO**	RG ← 1010000	**GOTO**	D		
D	**IF**	OB	**THEN**	F	**ELSE**	E
E	**IF**	PF	**THEN**	0	**ELSE**	D
F	**DO**	RG ← 1000000	**GOTO**	10		
10	**DO**	RG ← 1000100	**GOTO**	11		
11	**IF**	OB	**THEN**	11	**ELSE**	12
12	**DO**	RG ← 1000000	**GOTO**	C		
13	**DO**	RG ← 1000101	**GOTO**	14		
14	**IF**	T1	**THEN**	18	**ELSE**	15
15	**IF**	V	**THEN**	16	**ELSE**	14
16	**CALL**		1A	**RET TO**		17
17	**IF**	CV	**THEN**	7	**ELSE**	18
18	**DO**	RG ← 1001100	**GOTO**	19		
19	**IF**	T2	**THEN**	0	**ELSE**	19

1A	**CALL**		1F (0)	**RET TO**			1B
1B	**CALL**		1F (1)	**RET TO**			1C
1C	**CALL**		1F (2)	**RET TO**			1D
1D	**CALL**		1F (3)	**RET TO**			1E
1E	**RET**						
1F	**CALL**		24 (L0)	**RET TO**			20
20	**CALL**		24 (L1)	**RET TO**			21
21	**CALL**		24 (L2)	**RET TO**			22
22	**CALL**		24 (L3)	**RET TO**			23
23	**RET**						
24	**IF**	CV	**THEN**	25	**ELSE**		26
25	**IF**	A_{Lk}	**THEN**	26	**ELSE**		27
26	**IF**	B_{Lk}	**THEN**	28	**ELSE**		29
27	**IF**	B_{Lk}	**THEN**	29	**ELSE**		28
28	**DO**	CV ← 1	**GOTO**	2A			
29	**DO**	CV ← 0	**GOTO**	2A			
2A	**RET**						

- Codage des variables de test : nous divisons les variables d'entrée en deux types : variables de test du système et variables de test de comparaison ; nous allons les distinguées par le bit i_3. le codage des huit variables de test du système (V,PF,PO,OB,CV,T_1,T_2,D) et des deux variables de test de comparaison (A_{Lk} ,B_{Lk}) sera comme suit :

Variables	V	PF	PO	OB	CV	T_1	T_2	D
Code($i_2i_1i_0$)	000	001	010	011	100	101	110	111

$A_{Lk} \rightarrow i_4 = 0$, $B_{Lk} \rightarrow i_4 = 1$.

- Codage des paramètres : il y' a deux paramètres L et K, chacun peut prendre quatre valeurs, on les code comme suit :

K	0	1	2	3
CODE(k_1,k_0)	00	01	10	11

L	0	1	2	3
CODE(L_1,L_0)	00	01	10	11

Un bit supplémentaire est nécessaire pour distinguer entre les deux registres intermédiaires des deux paramètres L et K :

RZ1=REG[L] → P = 0, RZ2=REG[K] → P = 1.

❖ Enfin un bit j_0 pour différencier entre les deux registres REG[CV] et REG(sortie) :

REG=REG(sortie) → j_0 = 0, RZ3=REG[CV] → j_0 = 1.

Ainsi le format final des micro-instructions sera le suivant :

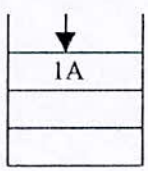
IF...	01	i_4 i_3 i_2 i_1 i_0	ADR1	ADR0
DO...	10	- - - - -	ADR+	OUT
CALL...	00	P l_1 l_0 k_1 k_0	ADSP	ADRT
RET.	11	- - - - -	-------	-------

Réalisation matérielle et chronologie d'exécution :

Le système microprogrammé destiné à exécuter notre microprogramme doit être compatible avec le format des micro-instructions que nous avons imposé ; le système que nous proposons à la figure(III.12) répond à cette condition. L'exécution de ce programme suit la chronologie suivante :

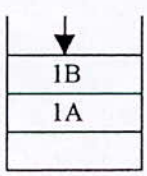
❖ De l'adresse 0 à l'adresse 4 exécution normale des instructions de test et d'affectation ; le bit i_3 sélectionne le MUX2 (variable de test du système) le bit j_0 sélectionne le registre de sortie.

❖ A l'adresse 5 appel au sous programme de l'adresse 1A et charge la pile par l'adresse de retour 6 (PUSH = 1).



PILE

❖ A l'adresse 1A appel au sous programme de l'adresse 1F et mémorisation de l'adresse 1B dans la pile ; le registre RZ2 = REG[L] se charge par la valeur L_1L_0 = 00.

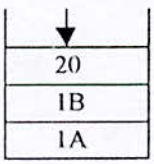
PILE


Microprogramme binaire				Hexadécimal			
Adresse	OP	$i_4\ i_3\ i_2\ i_1\ i_0$ / J_0 / $P\ l_1\ l_0\ k_1\ k_0$	ADR 1 / ADR + / ADSP	ADR 0 / OUT / ADRT	ADR	ROM = [ADR]		
000000	10	- - - - 0	000001	0000010	0	40	01	02
000001	01	- 1111	000010	- 000001	1	2F	02	01
000010	10	- - - - 0	000011	1000100	2	40	03	44
000011	01	- 1101	000000	-000100	3	2D	00	00
000100	01	- 1000	000101	-000011	4	28	05	03
000101	00	- - - - -	011010	-000110	5	00	2A	06
000110	01	- 1100	000111	-010011	6	2C	07	13
000111	10	- - - - 0	001000	1100000	7	40	08	C0
001000	01	- 1010	001001	-001000	8	2A	09	08
001001	10	- - - - 0	001010	1000100	9	40	0A	44
001010	01	- 1101	001010	-001011	A	2D	0A	0B
001011	01	- 1011	010000	-001100	B	2B	20	0C
001100	10	- - - - 0	001101	1010000	C	40	0D	50
001101	01	- 1011	001111	-001110	D	2B	0F	0E
001110	01	- 1001	000000	-001101	E	29	00	0D
001111	10	- - - - 0	010000	1000000	F	40	10	40
010000	10	- - - - 0	010001	1000100	10	40	11	44
010001	01	- 1011	010001	-010010	11	2B	11	12
010010	10	- - - - 0	001100	1000000	12	40	0C	40
010011	10	- - - - 0	010100	1000101	13	40	14	45
010100	01	- 1101	011000	-010101	14	2D	18	15
010101	01	- 1000	010110	-010100	15	28	16	14
010110	00	- - - - -	011010	-010111	16	00	1A	17
010111	01	- 1100	000111	-011000	17	2C	07	18
011000	10	- - - - 0	011001	1001100	18	40	19	4C

011001	01 - 1110	000000	-011001	19	2E	00	19
011010	00 000 - -	011111	-011011	1A	00	1F	1B
011011	00 001 - -	011111	-011100	1B	04	1F	1C
011100	00 010 - -	011111	-011101	1C	08	1F	1D
011101	00 011 - -	011111	-011110	1D	0C	1F	1E
011110	11 - - - - -	- - - - - -	- - - - - -	1E	60	00	00
011111	00 1 - - 00	100011	-100000	1F	10	23	20
100000	00 1 - - 01	100011	-100001	20	11	23	21
100001	00 1 - - 10	100011	-100010	21	12	23	22
100010	00 1 - - 11	100011	-100011	22	13	23	23
100011	11 - - - - -	- - - - - -	- - - - - -	23	60	00	00
100100	01 - 1100	100101	-100110	24	2C	25	26
100101	01 00 - - -	100110	-100111	25	20	26	27
100110	01 10 - - -	101000	-101001	26	30	28	29
100111	01 10 - - -	101001	-101000	27	30	29	28
101000	10 - - - - 1	101010	- - - - - -1	28	41	2A	01
101001	10 - - - - 1	101010	- - - - - -0	29	41	2A	00
101010	11 - - - - -	- - - - - -	- - - - - -	2A	60	00	00

- A l'adresse 1F appel au sous programme de l'adresse 24 et empile l'adresse 20 dans la pile ; le registre RZ1 = REG[k] se charge par la valeur $k_1k_0 = 00$.

PILE



- A l'adresse 24 test de la variable CV, MUX2 est sélectionné.

- A l'adresse 25, test de la variable A_{00} ; cette variable est sélectionnée par les sorties des registres RZ1 et RZ2 qui sont à la valeur $L_1L_0k_1k_0 = 0000$ et par le bit $i_4 = 0$; le MUX1 est sélectionné par le bit $i_3 = 0$.

- A l'adresse 26, test de la variable B_{00} ; cette variable est sélectionnée par $L_1l_0k_1k_0 = 0000$ et par $i_4 = 1$; (ADR+ = 28 ou 29).

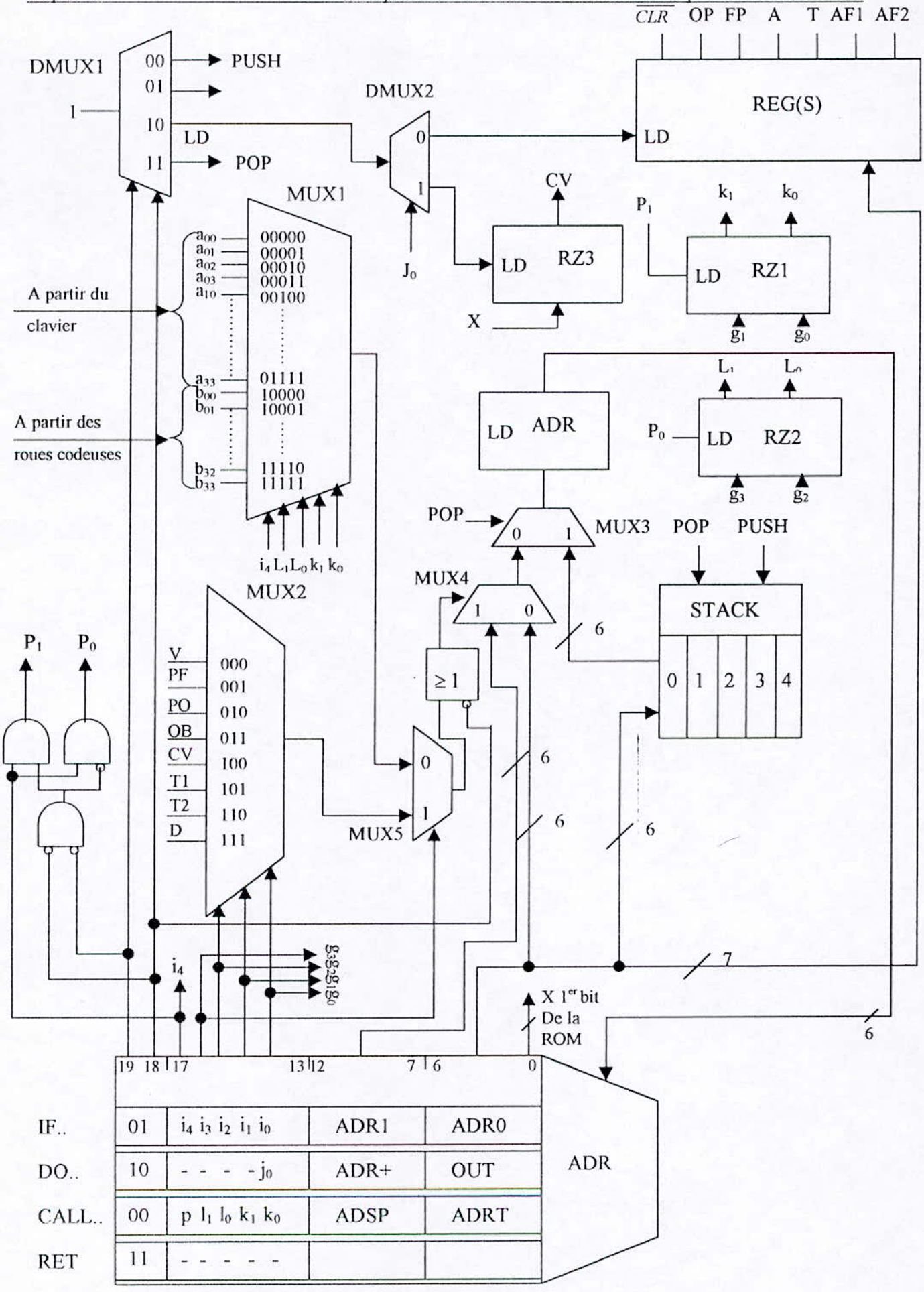
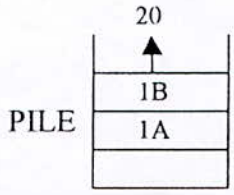


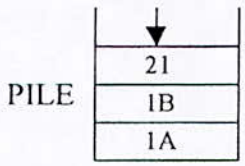
FIG.III.12 :schémas électrique

- A l'adresse 28 (29), sélection du registre RZ3 qui se charge par la valeur CV = 1 (CV = 0) ; ADR+ = 2A.

- A l'adresse 2A, restitution de l'adresse 20 stockée dans la pile (POP = 1) ; (ADR+ =20).



- A l'adresse 20, appel au sous programme de l'adresse 24 et empile l'adresse 21 dans la pile, le registre RZ1 se charge par la valeur $k_1k_0 = 01$; (ADR+ = 24).



- A l'adresse 24, test de la variable A_{01}; cette variable est selectionnée par les sorties des registres RZ_1 et RZ_2 qui valent $L_1L_0k_1k_0 = 0001$ et par $i_4 = 0$.....etc.

XI. Simulations et interprétations:

XI.1 Introduction:

Pour faire la simulation de notre système, on a utilisé le logiciel "Electronics workbench" qui présente des inconvenients et des prévilèges. Parmi les inconvenients majeurs qui ne nous ont pas facilté notre tache on cite: la non disponibilité des composants mémoires, ce qui nous a obligé à simulé l'EPROM par des DMUXs pour assurer l'adressage et des diodes pour simuler le programme de l'EPROM, on nous inspirons de l'une des méthodes de fabrication des ROMs et des EPROMs (voire annexe), ce qui a rendu notre système trés encombrant et c'est pour cette raison que nous avons choisi de simuler le circuit corespondant au premier cas ou la comparaison se fait par hard. L'autre inconvenient est la limite des variables qui peuvent ètre simulé en mème temps par les outils de mésures disponibles dans le logiciel qui vaut huit au maximum, or notre système possède un bus de sortie (ADR+, ADR0, ADR1, OUT, ADSP, ADRT) d'une taille de 12 bits ce qui nous a obligé à l'executer en deux parties: la preière donnant les sorties du registre d'adresse future (5 bits) et la deuxième les sorties du registre de sortie (7 bits); pour les deux cas en prend le premier bit de l'EPROM, qui corespond au code instruction, comme reference.

Malgré ces inconvénients l'*Electronic workbench* demeur un puissant logiciel de simulation électronique du fait de la diversité des composants et des appareils de mesure qu'il offre, et aussi du fait de la facilité et la simplicité de son utilsation.

XI.2. Simulation et interprétation :

Pour la simulation on a utilisé l'outil de mesure « Logic analyzer » qui fait l'analyse de huit canaux d'entrée en les représentant sous forme d'ondes carrées en fonction du temps, il dans aussi leur représentation binaire et hexadécimale à chaque instant.

On va faire quatre exemples qui correspondent à quatre cas de variables d'entrée ou de test qui englobent pratiquement les différentes situations du système et les différentes lignes du microprogramme

Notons que la fréquence du travail du système est de 1MHz pour tout les exemples. Notons aussi que le premier bit de chaque figure représente le code instruction qui vaut 1 pour l'instruction **DO...**, et 0 pour l'instruction **IF...**

❖ **1er cas :**

Dans ce cas on va vérifier que tant qu'aucun chiffre n'est introduit par le clavier le système reste à l'état d'attente. Pour cela on pose la variable D égale à 0.

La figure(III.13-a)montre bien que le système évolue de l'adresse initiale (ADR = 0) vers l'état caractérisée par l'adresse (ADR = 1) et il y reste tant que D = 0 , et le registre de sortie se charge par la valeur 0000010 et y reste.(voir figure (III.13-b).

❖ **2eme cas :**

Dans cet exemple encore simple comme le premier, on envisagé ou une personne a introduit une série de chiffres par le clavier, mais qui n'a pas voulu la valider, en posant D =T1=1. Donc d'après le cahier des charges et le microprogramme, le système doit revenir à l'état initial puisque le code introduit n'a pas été validé avant les 16 secondes après l'introduction du premier chiffre.

Les figures (III.14-a) et (III.14-b) montrent bien que le système après détection du premier chiffre introduit, déclenche une temporisation et revient à l'état initial après les 16 secondes : en passant de l'état initial (adresse 0 et registre de sortie égale à 0000010) à l'adresse 1 puis l'adresse 2 ou il charge la sortie par 1000100 et passe l'adresse 2 et revient à l'adresse 0.

❖ **3eme cas :**

Dans cet exemple on est allé un peut plus loin en affectant aux variables de test les valeurs suivantes : D = T2 = V = 1, CV = T1 = 0, ce qui nous a permit d'aller plus loin dans la vérification de la validité du microprogramme et ainsi de vérifier le bon comportement du système pour des situations plus diverses et plus avancées.

Les figures (III.15-a) et (III.15-b) montre effectivement que le système à l'état initial charge la sortie par 000010, passe à l'adresse 1 puis à l'adresse 2 ou il charge la sortie par 1000100, ensuite passe à l'adresse 3, l'adresse 4, l'adresse 5 puis l'adresse 12 ou il charge la sortie par 1000101et passe à l'adresse 13, l'adresse 14, l'adresse 15 et en suite l'adresse 16 ou il charge la sortie par 1001100 et passe à l'adresse 17, ou il revient à l'état initial (adresse 0 et sortie égale à 0000010).

Cet exemple nous a permit de vérifier le bon fonctionnement des options de temporisation .

❖ **4eme cas :**

Dans cet exemple, on a introduit les variables de test restantes à savoir PO, PF et OB en posant V = D = CV = PO = PF = 1 et OB = T1 =0, donc nous avons envisagé le cas ou le code introduit est valide.

Les figures (III.16-a) et (III.16-b) montrent bien que le système, de l'adresse 0 (état initial) passe à l'adresse 1, l'adresse 2, l'adresse 3, l'adresse 4, l'adresse 5 puis l'adresse 6 ou il charge la sortie par 1100000 et passe à l'adresse 7puis l'adresse 8 ou il charge la sortie par 1000100 et passe à l'adresse 9, l'adresse A puis l'adresse B ou il charge la sortie par 1010000 et passe à l'adresse C, l'adresse D puis l'état initial et charge la sortie par 0000010.

Dans cet exemple, on a bien vérifié le bon fonctionnement des options de fermeture, d'ouverture de la porte et de la détection des obstacles.

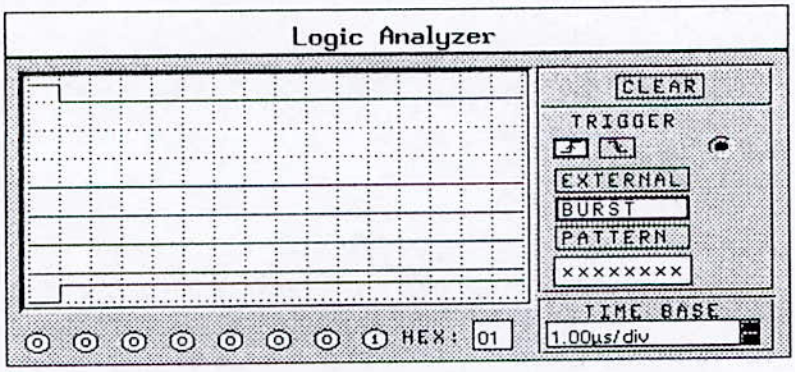


FIG.III.13-a

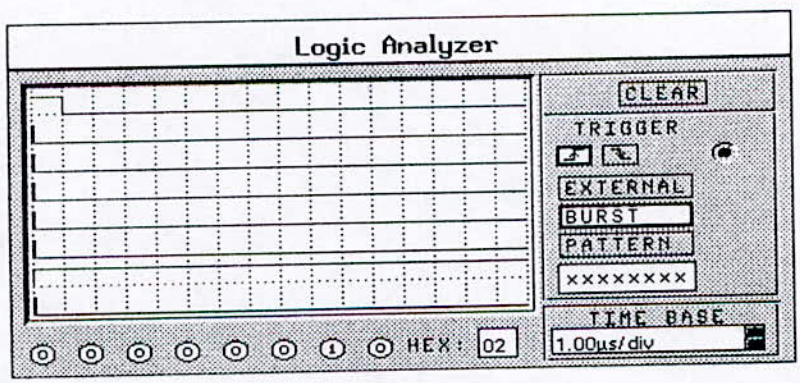


FIG.III.13-b

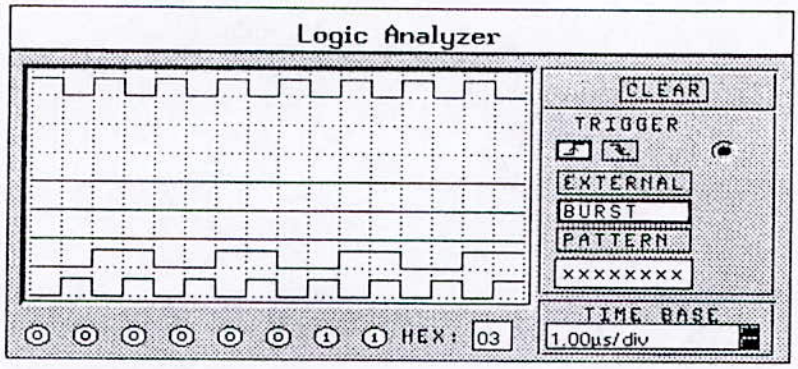


FIG.III.14-a

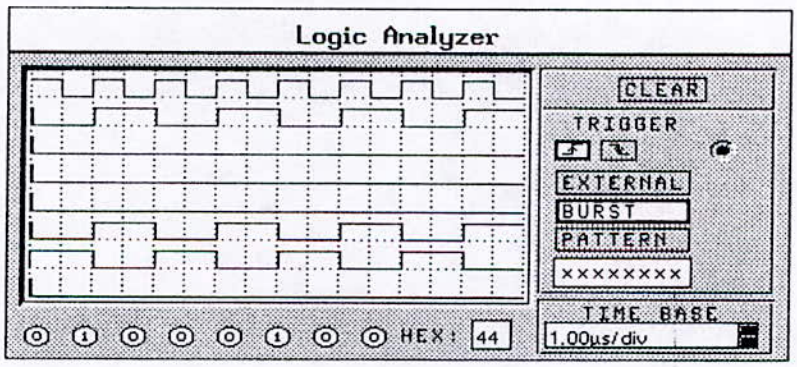


FIG.III.14-b

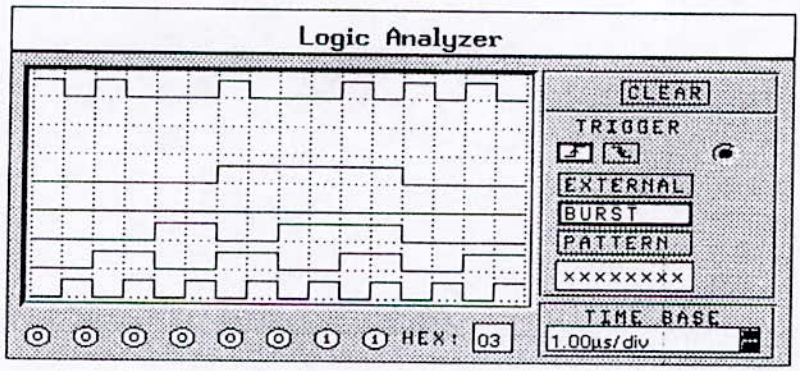


FIG.III.15-a

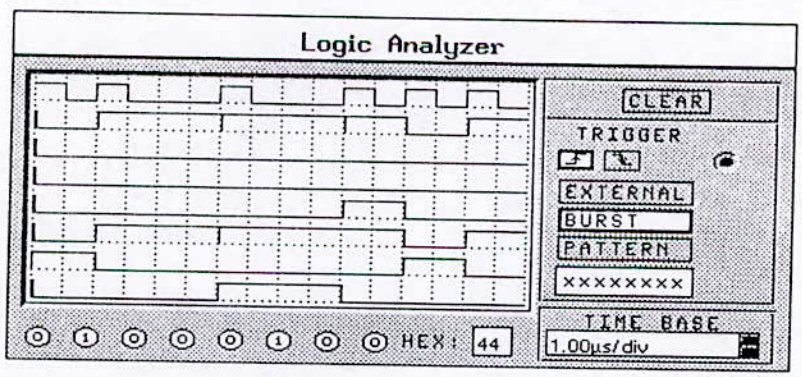


FIG.III.15-b

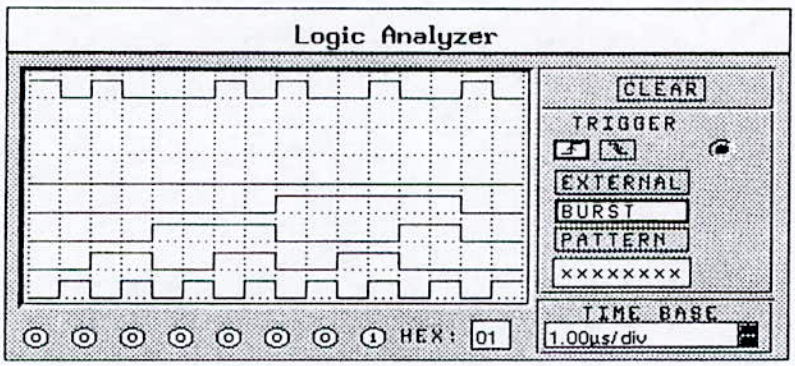


FIG.III.16-a

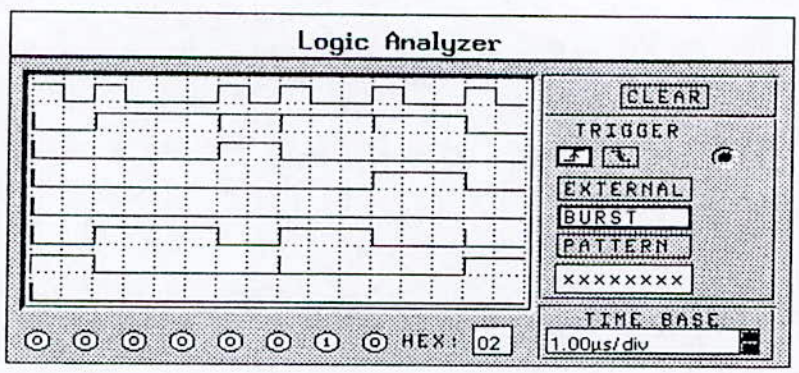


FIG.III.16-b

XII. Conclusion :

On a vu dans ce chapitre l'application de la méthode microprogrammée étudiée dans les deux chapitres précédents, à la conception d'un automate de commande d'une porte de sécurité à code. Nous avons commencer par établir un cahier des charges et un organigramme répondant à ses exigences ; étape nécessaire dans tous processus de conception, puis nous avons dressé le DDB qui n'est qu'une traduction en binaire de l'organigramme qui nous a permit la rédaction du microprogramme mnémonique et binaire. Pour terminer l'étude théorique, nous avons proposé un schéma électrique basé sur une unité centrale qui est l'interpréteur des micro-instructions, capable d'exécuter le microprogramme.

Pour cette étape nous avons envisagé deus cas :

* Cas où la comparaison se fait par *hard* à l'aide d'un comparateur.

* Cas où la comparaison se fait par *soft* qui nous a permit d'économiser en matériel au prix d'une augmentation de la taille du microprogramme.

Pour des entrées imposées, les résultats de simulations qui donnent l'adresse futur et la sortie étaient en parfait accord avec l'étude théorique.

CONCLUSION GENERALE

Conclusion générale :

Notre travail a été consacré sur l'application de la méthode microprogrammée dans la conception d'automates séquentiels.

Au premier chapitre, nous avons vu qu'une machine séquentielle peut être décrite par plusieurs méthodes : tabulaire, graphique ou par microprogramme. Cette dernière résulte d'un diagramme de décision binaire où chaque état de diagramme représente une adresse du microprogramme.

Au deuxième chapitre, nous avons étudier avec plus de détail la méthode microprogrammée, où nous avons vu qu'un microprogramme qui ne contient que les deux primitives de spécification , qui sont les deux micro-instructions de test et d'affectation, était linéaire, alors que la présence de blocs répétitifs dans le microprogramme nous a obligé d'introduire les notions de sous microprogrammes et procédures (microprogramme non-linéaire) qui nécessite une solution matérielle à savoir : la Pile.

Encore une fois le format des micro-instructions était large ; la combinaison d'un nouveau concept logique, le microprogramme incrémenté, et d'un nouveau opérateur matériel, le séquenceur, nous a permet de réduire la largeur des micro-instructions ; ce gain en largeur sera compensé par une petite perte en profondeur du microprogramme.

Au troisième chapitre nous avons appliqué la méthode microprogrammée à la conception d'un automate de commande d'une porte de sécurité à code ; nous avons étudier les deux cas : cas où la comparaison se fait par hard à l'aide d'un comparateur, et le cas où elle se fait par soft à l'aide d'un sous microprogramme de comparaison, ce dernier cas nous a permet d'économiser un peu du matériel. Nous avons suivi toute les étapes de cette méthode en commençant par l'élaboration d'un cahier des charges répondant à nos exigences, l'organigramme, le diagramme de décision binaire, le microprogramme mnémonique et binaire et, enfin, nous avons proposé un schéma électrique compatible avec le format des micro-instructions et capable d'exécuter notre microprogramme.

Pour cette première partie du troisième chapitre nous avons vu la bonne correspondance entre le matériel, qui était peu encombrant et pas coûteux, et le logiciel, qui était de sa part facile à rédiger.

La deuxième partie de ce chapitre comprenait les résultats de simulation sur ordinateur de notre système, ces résultats contenaient les valeurs des registres de sortie et d'adresse future ; qui étaient en accord parfait avec l'étude théorique, où nous avons vu la facilité et la simplicité qu'offre cette méthode aux concepteurs d'automatismes ; puisqu'elle n'exige pas la maîtrise d'un jeu d'instruction à l'instar des automatismes conçus à base de microprocesseurs.

La conception d'automatismes par la méthode microprogrammée recouvre l'art et la technique des transformations et équivalences entre matériel : système logique, et logiciel : les microprogrammes.

ANNEXE

ANNEXE :

Système anti-rebonds :

Pour remédier au problème causé par le phénomène de rebondissement des contact électriques des touches du clavier (figure 1), on a utilisé une bascule monostable caractérisée par un temps de propagation τ_p (figure 2).

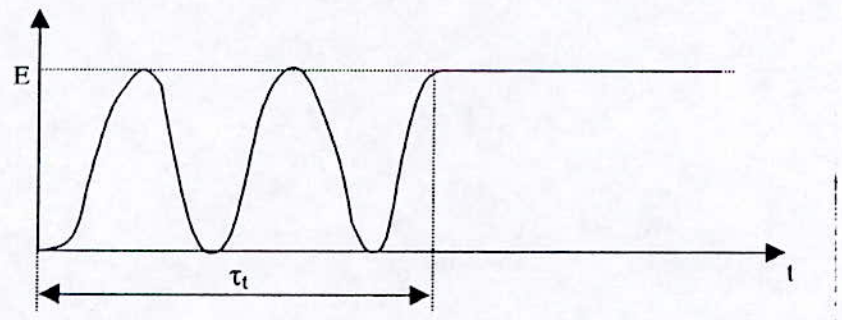


FIG.1

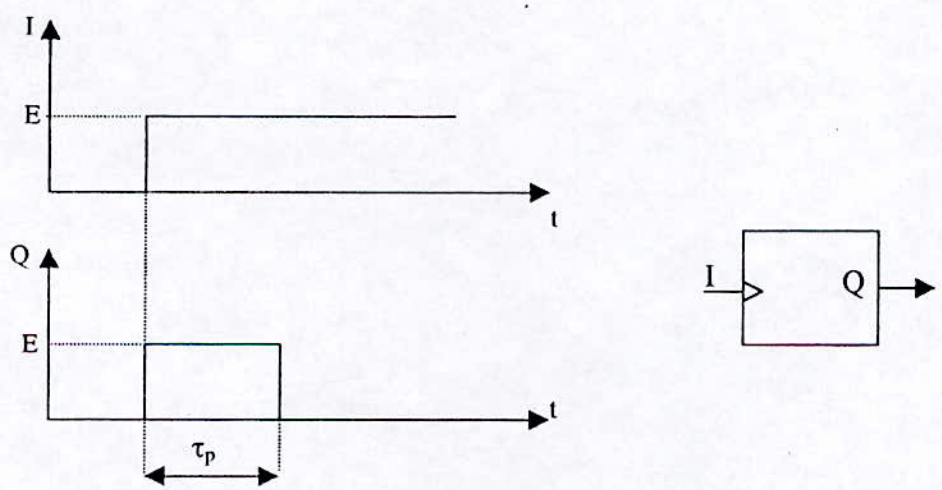


FIG. 2

Donc on doit assurer que $\tau_p > \tau_t$, en général une valeur de 20ms pour τ_p est suffisante.

Circuiterie de l'alarme sonore :

Le circuit de commande de l'alarme sonore est constitué de deux astables en cascade (figure 3) ; l'astable en aval est commandé par le signal de déclenchement de l'alarme A, cet astable est configuré pour générer une fréquence de 1Hz. Le second génere une fréquence de 1KHz, et on obtient le signal de l'alarme représenté par la figure 4.

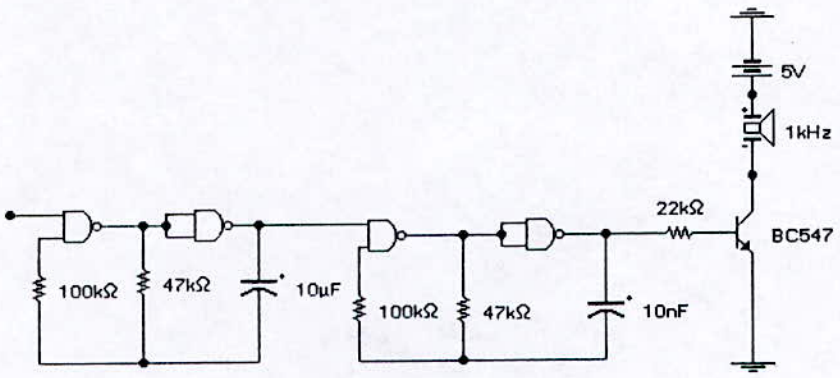


FIG.3

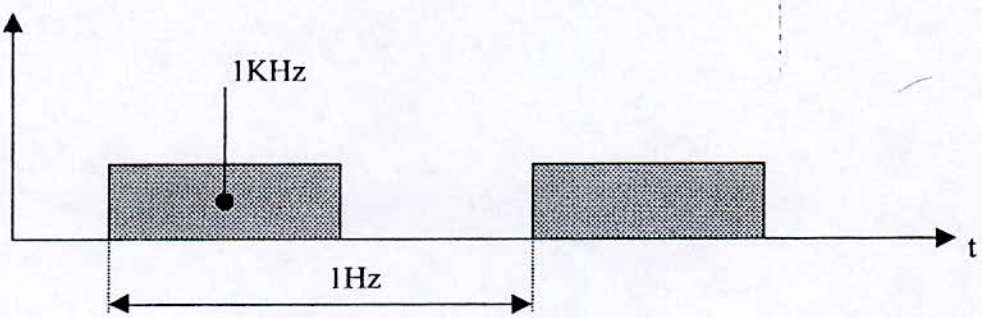


FIG. 4

Gestion du moteur :

Pour commander l'ouverture et la fermeture de la porte, on utilise la commande d'un moteur « marche avant/ marche arrière ». Pour cela on utilise un moteur à courant continu à aimant permanent, dont l'inversion du sens de rotation s'obtient en inversant la polarité de la tension appliquée à ses bornes. Cela peut être réalisé manuellement à l'aide d'un inverseur, mais dans les systèmes de commande automatique ce qui est notre cas, c'est plutôt effectué à l'aide de transistors. Un circuit type est illustré schématiquement dans la figure 5.

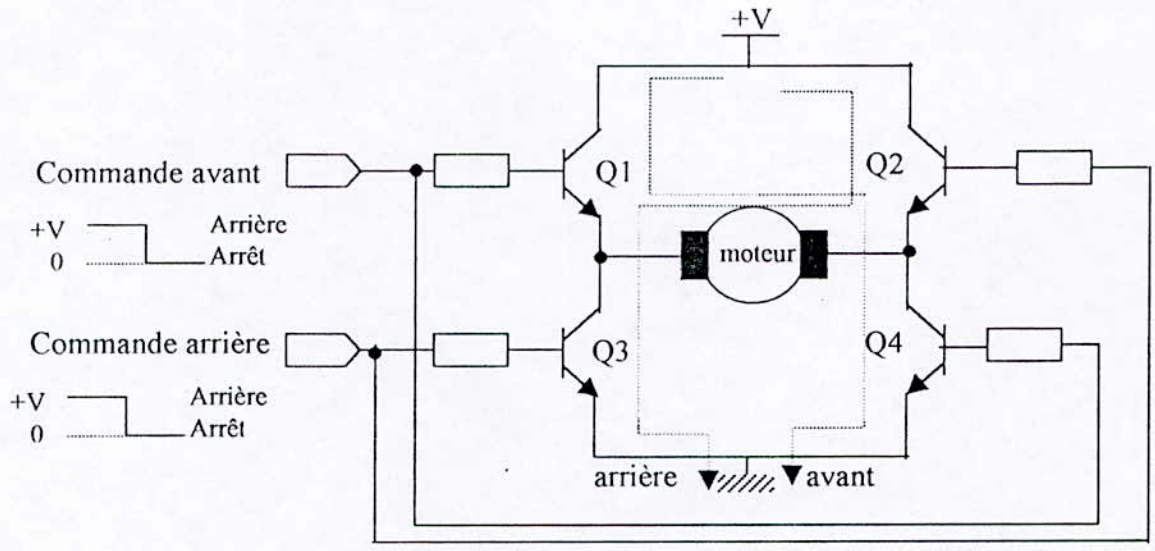


FIG.5

Un signal marche avant commande let transistors Q1 et Q4, aiguillant le courant comme le montre la figure 5. Le signal de commande marche arrière fait conduire les transistors Q2 et Q3 pour faire passer le courant dans le moteur en sens inverse.

Dans notre cas, la commande de l'ouverture ou de la fermeture de la porte se fait en associant les signaux PO et PF à l'une des commande avant ou arrière.

Conception des mémoires mortes :

Parmi les différentes méthodes de conception des mémoires mortes, on cite la méthode ou la programmation des mots dans ces mémoires se fait par des diodes. Par exemple pour une mémoire de 4 × 4 ayant les mots suivants : 0001, 0110, 1110 et 1001, nous aurons la mémoire représentée par la figure 6, où les lignes représentent l'adresse et les colonnes le mot correspondant à cette adresse.

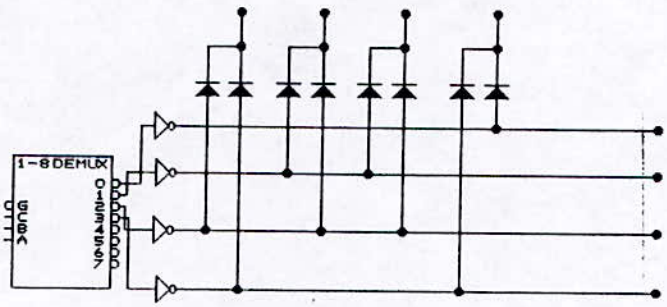


FIG.6

BIBLIOGRAPHIE

Bibliographie :

[1] Logique transitionnelle microprogrammée, conception des automates industriels / J. Florine- Paris, inter édition, 1983-677P.

[2] Pratique des circuits logiques /Jean- Michel. Bernard, J. Hugan,...4ème édition. Eyrolles, 1990-453P.

[3] Circuits/ Steve CIARCIA, traduction de l'Amérícan- Anckland ; Mc Graw- Hill, 1985-1986-2t, 236P. traduction de « Ciarcia's circuit cellar ».

[4] Machines séquentielles / Jacques. Zahnd-5t- Saphorin (Suisse) : Georgie, 1980.

[5] Systemes microprogrammés /Daniel Mange- Une introduction au Magiciel, presse polytechnique et universitaires, ROMANDES, 1990.

[6] Analyse et synthèse des systemes logiques, Traité d'electricité –Vol.5 / Daniel Mange, presse polytechnique et universitaires, ROMANDES, Lausanne, 1985.

[7] Design of logic systems/ D. Lewin, Berckshire(G. R.) : E.L.B.S, Van Nostrand Reinheld, 1985.-578p.-(Coll « E.L.B.S. »).

[8] Logique séquentielle/ Jean LAGASSE ; La coll. De M. Courvoisier, J.-P.Richard,...-3ème éd.- Paris : Dunod, 1976-VI- 81P.

[9] Logique combinatoire et séquentielle : maitrise d'E.E.A,...., J. LAGASSE,.... ; avec la collab. De J. ERCEAU,...-2ème éd.-. Paris : Dunod, 1971,-IX- 187-Bibliogr-

[10] Introduction aux automatismes industriels ; Grafcet et logique électronique avec exercices et solutions/ Y. LECOURTIER ; ...B. Saint-jean ;....-Paris :Masson.1989.-232P.