N° d'ordre :

THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour l'obtention du titre de

Docteur de l'Université

par

L. Teixeira de Carvalho

Ingénieur Industriel

Belgique

CONTRIBUTION A L'ETUDE D'UN SYSTEME INTERACTIF A MICROPROCESSEUR, CONDUISANT DU CAHIER DES CHARGES A LA CARTE DE CIRCUIT IMPRIME.

Soutenue le 29 juin 1981 devant la Commission d'Examen

Membres du Jury : MM. P. VIDAL — Président
J. M. TOULOTTE — Rapporteur
G. MANESSE — Examinateur
G. FINCKEN — Invité

AVANT - PROPOS.

Nous remercions Monsieur P. VIDAL, d'avoir bien voulu accepter la présidence de notre Jury de Thèse.

Nous sommes très heureux de pouvoir exprimer ici notre gratitude à Monsieur J.M. TOULOTTE, qui nous a suivi et guidé durant cette étude.

Nous remercions également Messieurs les Membres du Jury, d'avoir bien voulu siéger à cette Commission d' Examen.

T A B L E D E S M A T I E R E S.

I N T R O D U C T I O N G E N E R A L E

Le projet contenu dans l'énoncé du titre de ce travail semble à première vue quelque peu démesuré. A la reflexion pourtant, l'interactivité apparait comme l'élément devant en permettre la réalisation. Elle devrait en réduire la complexité et le volume tout en améliorant l'efficacité.

Mais, afin de préciser exactement le point de départ du travail, définissons le contenu du cahier des charges:

a) le nom du circuit étudié
b) la définition des signaux d'entrée
c) la définition des signaux de sortie
d) les contraintes relatives à la réalisation
e) les relations devant exister entre les signaux de sortie et ceux d'entrée.

L'ensemble de cette étude sera divisée en quatre parties:

1- Un "éditeur de description fonctionnelle"
devant permettre de passer de l'énoncé du problème, contenu dans le cahier des charges, à la description fonctionnelle de la solution dans un langage de type PHPL (Parallel Hardware Processing Language).

2- Un Simulateur
devant permettre de s'assurer que la description produite en "1" réponde correctement au problème posé.

3- Un "Traducteur Matériel"
devant transposer la description fonctionnelle en schéma matériel. Cette transposition implique des choix de technologies et cela à plusieurs niveaux. Notons que nous ne considérons ici qu'une transposition en circuit électronique

4- Un "Concepteur de Carte imprimée"
devant produire le dessin de la carte de circuit imprimé correspondant au schéma matériel obtenu précédement.
Cette partie comportera deux phases:

1) le placement des composants sur la carte
2) la recherche des interconnexions

Note: dans le cas du cablage matricé (Wire Wrapping) la deuxième phase est simplement remplacée par la liste des interconnexions à réaliser.

L'interactivité étant le fil conducteur de cette étude, les quatre parties précédentes seront regroupées dans un programme principal que nous appelerons "Programme d'Aide à la Conception Matérielle".

Ainsi, à la fin de l'exécution de chacunes des phases du travail, il y aura retour au programme principal qui rendra l'initiative au concepteur. Ce dernier pourra alors appeler au choix n'importe lequel des programmes particuliers:édition, simulation, transposition matérielle ou conception de carte imprimée.

Cette procédure nous semble favoriser le dialogue machine-concepteur, ce qui est le but de l'interactivité.

Dans l'étude de chacune des parties nous nous efforcerons d'éviter de faire exécuter à la machine un travail pour lequel elle n'est pas apte.

Nous lui confierons au contraire tout ce qu'elle fera vite et bien afin d'en décharger le concepteur.

Il est clair que, dans ces conditions, le système sera plus simple et plus efficace.

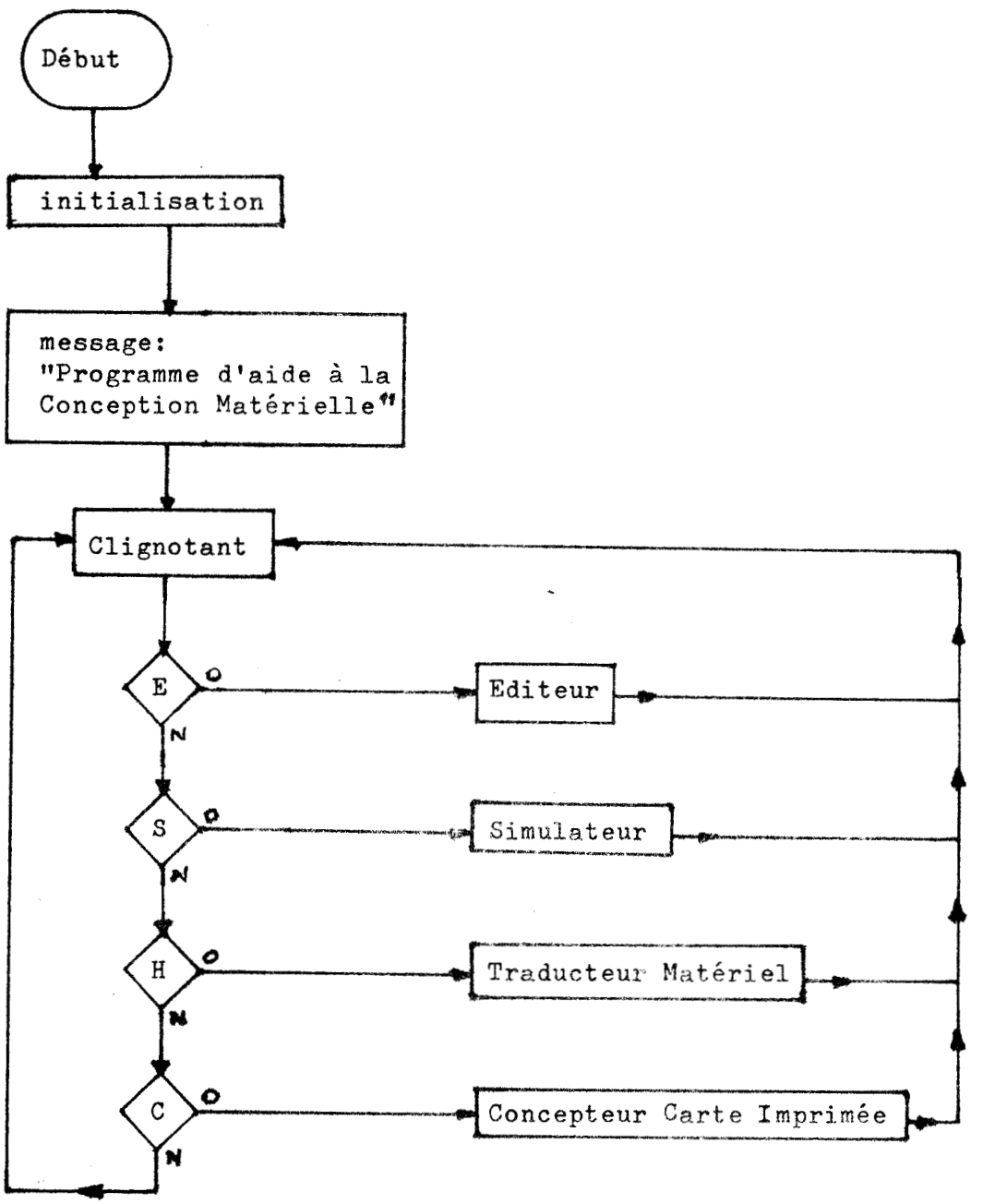
Afin de décider à bon escient du travail que l'on confiera à la machine, il convient de se souvenir qu'elle possède les qualités suivantes:

- mémorisation de quantités relativement importantes d'informations → banque de données.

- rapidité d'exécution de travaux de routine mettant en oeuvre des techniques simples et répétitives.

Par contre le concepteur aura lui une vue plus globale des problèmes.

- Ordinogramme du Programme principal:



C H A P I T R E I

E D I T E U R D E D E S C R I P T I O N F O N C T I O N N E L L E.

I. 1. INTRODUCTION.

L'éditeur fournira au concepteur, non seulement la possibilité d'introduire en machine, avec les facilités d'usages, la description fonctionnelle de la solution au problème posé, mais en plus, il lui apportera toute l'aide possible pour établir cette description.

I. 1.1. Introduction du texte

L'introduction du texte de la description en machine est, en fait, le rôle classique de l'éditeur.

Le texte, rentré à partir d'un clavier, devra être visualisé sur un écran et pourra être corrigé et altéré à volonté par l'opérateur.

Pour cela le clavier sera doté d'un certain nombre de touches de contrôles permettant d'exécuter les commandes suivantes:

1) contrôle du curseur

2) effacement d'une ou plusieurs lignes

3) insertion d'une ou plusieurs lignes à un endroit donné d'un texte

4) mémorisation du texte visualisé dans un tampon

Notons également qu'un caractère clignotant indiquera, sur l'écran, la position actuelle du curseur.

I. 1. 2. Aide à la conception

Nous avons dit, que dans notre cas, il incombait également à l'éditeur d'apporter toute l'aide nécessaire pour établir la description fonctionnelle solution du problème.

Nous avons dit aussi, que la machine était apte à mémoriser des quantités relativement importantes d'informations et à exécuter rapidement des travaux de routine simples.

Nous n'envisagerons toutefois pas ici de lui demander d'appliquer des techniques de synthèse de circuits combinatoires ou séquentiels. Nous estimons en effet que souvent la solution au problème posé fera appel, comme éléments de base, à des modules auxquels ces techniques seront difficilement applicables.

Douglas LEWIN (Brunel University) (33) estime quant a lui que "la théorie a été (et est toujours) dépassée par la technologie et qu'un problème majeur existe maintenant du à l'absence d'une théorie de conception appropriée au niveau des sous-systèmes; ....La situation devenant encore plus critique maintenant que des circuits électroniques programmables comme microprocesseurs et micro-calculateurs sont utilisés en tant que composants".

A notre avis l'aide la plus efficace que la machine pourra apporter au concepteur, à ce stade du travail, sera de fonctionner en aide mémoire.

Aussi envisageons nous la procédure suivante:

1) après appel de l' "éditeur" celui-ci demande:
   1 - le nom du circuit à étudier
   2 - les noms des signaux d'entrée et de sortie

2) l' "éditeur" fournit alors la liste des fonctions et des modules dont il possède la description.

3) L'opérateur peut alors choisir certains modules ou fonctions et visualiser leur description. Il peut ainsi s'en servir pour élaborer sa solution au problème posé.

Notons qu'une fois la description terminée on peut demander à l' "éditeur" de vérifier si aucune entrée ou sortie n'a été oubliée ou connectée de façon incorrecte. Ce travail peut toutefois être confié au "simulateur".

I. 1. 3. Langages utilisés

Nous avons parlé jusqu'à présent de la liste des fonctions ou modules en mémoire et de leur description ainsi que de la description fonctionnelle de la solution. Il convient à présent d'envisager le ou les langages qui permettront de les exprimer.

Pour cela, il faudra avant tout faire la distinction entre ce qui devra être compréhensible à la fois par la machine et par le concepteur et ce qui pourra l'être du concepteur uniquement.

Ce qui devra être à la fois compris de la machine et du concepteur devra être écrit dans un langage précis, bien structuré et fort détaillé. Les langages convenant le mieux pour cela sont ceux que l'on appelle les "langages de description matérielle", (Hardware description languages:HDL), et dont nous parlerons en détail au point suivant.

Le défaut de ce genre de langage est de ne pas donner au concepteur une vue d'ensemble aisée des caractéristiques des fonctions ou modules décrits. Aussi sera-t-il souhaitable de donner à ce dernier des informations complémentaires, sur chaque fonction et module, qui lui permettent de les appréhender plus globalement et plus rapidement.

Ainsi, dans la liste des fonctions et modules, les noms devront être choisis autant pour leurs qualités mnémoniques que pour leur brieveté .

De plus, chaque description fonctionnelle, dans un langage "HDL", devrait être précédée d'une brève description en langage naturel (français, anglais).

La représentation, graphique ou semi-graphique, d'un "timing" bien choisi peut aussi, dans certains cas, aider à saisir plus rapidement et globalement certaines caractéristiques du module.

La possibilité de travailler avec des modules est également utile pour la conception de systèmes complexes.

I. 2. LES LANGAGES DE DESCRIPTION "MATERIELLE" (HDL) (28,34,48 et 55)

I. 2. 1. Introduction:

Les langages de description "matérielle" sont destinés à décrire un système digital, tant du point de vue de sa structure, que

de son fonctionnement. Une des utilisations les plus courantes des langages "HDL" est de s'en servir comme moyen d'entrée vers un simulateur, ce qui sera notre cas.

Les caractèristiques essentielles des langages "HDL" seront dès lors les suivantes:

- Il sera important de pouvoir décrire aisément le parallélisme d'un système digital
- Il sera également souhaitable de pouvoir séparer la description de la structure logique du système de celle de la partie correspondant au contrôle.
- Le langage devrait pouvoir servir à tous les niveaux de description pour ne pas être obligé d' utiliser plusieurs langages pour couvrir les différents stades de la conception du système.
- Il sera utile également de pouvoir introduire des modifications dans les descriptions sans difficultés, surtout lorsque l'on travaille en mode interactif.
- Enfin le langage devra pouvoir s'apprendre et se retenir facilement.

Peu de langages "HDL" ont réussi à être raisonnablement bien connus. Le langage "CDL" de Y. CHU (10) apparait en 1965 et est fort utilisé, particulièrement en Europe. Le langage "DDL" (12) a été introduit en 1967 par D.L. DIETMEYER et son étudiant J.R. DULEY et est largement utilisé dans l'industrie U.S. Le langage "ISP" (47) et le langage "PMS" (48) sont introduit par C.C. BELL et A. NEWELL en 1971 et sont bien connus de divers constructeurs et universités. Le langage "AHPL" (24) a été introduit également en 1971 par F. HILL et G. PETERSON. Il y a de nombreux autres langages, nous en donnons

ci-dessous une liste exemplative (55).

En fait ces langages sont fort similaires les uns aux autres même s'ils utilisent un symbolisme différent. Ils ont toujours été développés pour être implantés sur des machines existantes et différentes, ce qui a eu pour effet de provoquer leur prolifération abusive.

Langage	Adapté de	Implanté sur machine	Langage d'implantation
ACDL	-	-	-
AHPL	APL	CDC-6400	SNOBOL
		DEC-10	Fortran
ALERT	APL	IBM 7094	-
APDL	ALGOL	CDC-G20	ALGOL-60
APL	-	plusieurs	Assembleur
APL*DS	APL	-	-
CASD	PL/1	IBM 360	PL/1
CASSANDRE	ALGOL	IBM 360, 370	Assembleur
CDL	ALGOL	IBM 370	Fortran Assembleur
CSL	ALGOL	IBM 370/155	BCPL
DDL	-	Harris 6024/3	-
DIGITEST II	P1/1	-	-
	Réseaux de Petri		
DIDL	-	-	-
DSDL	DDL	IBM 360	XPL
FLOWWARE	ordinogramme	IBM 360/50	PL/1
FST	-	IBM 360	Fortran IV
GLIDE	-	-	-
HARGOL	ALGOL	-	-
HILO	-	ICL 1900	-

ISP			
ISPL	ALGOL	PDP-10	BLISS
ISPS			
LALSD	-	IBM 360/91	PL/1
		CDS 6400	SNOBOL
LASCAR	CASSANDRE	-	-
LCD	PL/1	-	-
LDT	RTL	Burroughs	ALGOL-58
LOGAL	RTL	UNIVAC 1108	Fortran IV
LOTIS	ALGOL	-	-
MDL	APL	-	-
MODEL/LINDA	-	-	-
OSM		ODRA-1305	PLAN
PMS	ALGOL	PDP-10	SNOBOL
RTL	-	CDC 1604	ALGOL
RTS I	ALGOL	Siemens 4004/151	Fortran
RTS II	RTS I	-	-
RTS III	CDL, RTS	-	-
SDL	RTL	-	-
SDL II	ALGOL	-	-
SDL	-	-	-
SMITE	-	CYBER 174	-
SSM	-	-	-
VDL	-	-	-
V-PMS	PMS	-	-

I. 2. 2. Les différents niveaux de langages "HDL":

Un système digital pouvant être décrit à plusieurs niveaux, il y aura aussi différents niveaux de langages "HDL" qui sont les suivants:

1) Le niveau le plus élevé est celui de l'algorithme

2) Le second niveau est le niveau PMS (Processor, Memory, Switch) qui décrit le système en sept types de composants: mémoire, connexion, contrôle, transducteur, opérateur et processeur.

3) Le niveau suivant est celui correspondant à la description à l'aide des instructions d'un "processeur".

4) Le quatrième niveau est appelé "tranfert de registre": les éléments du système décrit sont considérés comme étant des registres et le tranfert de données entre ces registres est soumis à certaines règles.

5) C'est le niveau logique qui décrit le système avec des portes et des bistables.

6) Le niveau le plus bas, c'est celui où les portes et les bistables sont décrits avec des éléments tels que transistors, résistances, etc.....

I. 2. 3. Catégories de langages "HDL":

Une description formulée dans un langage "HDL" se présentera sous la forme d'une suite de lignes et de textes que l'on appelera des "déclarations". Ces "déclarations" consisteront en une séquence de signes, de mots ou d'expressions qui traduisent une ou plusieurs actions matérielles qui devront être interpretées et réalisées par la machine.

Les langages "HDL" seront classés en deux catégories suivant l'ordre d'exécution des déclarations, relativement à l'ordre suivant lequel elles apparaissent dans la description:
Nous aurons ainsi: a) les langages "procéduriers"
b) les langages "non-procéduriers".

a) Les langages "non-procéduriers" sont ceux dont chaque "déclaration" contient une "étiquette" qui donne les conditions à remplir pour que l'action décrite par la "déclaration" soit exécutée.

Donc, pour ce type de langage, l'ordre suivant lequel les "déclarations" apparaissent dans la description n'impose pas l'ordre dans lequel doivent avoir lieu les actions.

b) Dans un langage "procédurier" les actions seront exécutées dans l'ordre d'apparition des "déclarations".

I. 2. 4. Choix d'un langage "HDL"

Nous dirons avant tout avec R.W. MARCZYNSKI et P. BAKOWSKI (35) qu'il est bien difficile pour un langage donné, d'en distinguer tous les bons et tous les mauvais aspects.

De plus, le nombre de langages proposés est tellement grand que le choix est ardu.

Nous nous sommes attachés à rechercher un langage permettant

1) de réaliser des descriptions modulaires

2) de décrire aisément les caractéristiques temporelles.

3) un niveau de description fonctionnelle.

Ces critères nous ont amenés à choisir le langage "PHPL" (Parallel Hardware Processing Language).

I. 3. LE LANGAGE "PHPL" (Parallel Hardware Processing Language).

I. 3. 1. Introduction

Le langage "PHPL" a été développé à l'Université Technique de Munich (5)

Ce langage permet la description du comportement fonctionnel de circuits logiques en tenant compte de la variable temps.

Cette description est construite à partir, d'une part de deux composants mémoires, les registres et les mémoires adressables et, d'autre part, d'éléments que nous appelerons bornes (borniers dans le cas d'un ensemble de bornes). Ces bornes constitueront le support aux variables d'entrées ou de sorties du circuit, ainsi qu'aux variables internes résultat de fonctions logiques opérées sur d'autres variables du circuit ou de délais introduits par rapport à ces variables.

Une description peut être subdivisée en modules.

I. 3. 2. Description des éléments du langage.

a) Alphabet

Le langage "PHPL" utilise comme alphabet les caractères suivants:

- les caractères alphanumériques A à Z et 0 à 9
- les lettres minuscules n, m, s
- la lettre grecque μ
- l'espace ou "blanc"
- les symboles ; : , . () = + ≠ ∩ ∪ / I ' * ← ↑ ↓ &

b) Chaînes de caractères

Les chaînes de caractères ou mots du langage seront de types suivant :

1 - Les variables

1° Les variables logiques (noms de signaux logiques)

- Ces variables seront exprimées par une chaîne de trois caractères alphanumériques maximum. Le premier caractère étant obligatoirement alphabétique et différent de T.

2° Les variables temporelles

- Ces variables seront exprimées par une chaîne de quatre caractères alphanumériques au maximum. Le premier caractère étant obligatoirement la lettre T.

2 - Les constantes

1° exprimant les valeurs de variables logiques:

1 ou 0

2° exprimant un temps ou les caractéristiques d'un composant:

- Ces constantes seront exprimées par un nombre décimal de quatre chiffres au maximum dont un chiffre décimal éventuel. La virgule compte alors pour un chiffre.

Notes : - 1 - Ce nombre sera suivi dans le cas ou il exprime un temps d'une des trois unités possible : ns, µs, ou ms.

- 2 - Des temps constants ou des intervalles de temps pourront être défini de manière symbolique . Le nom symbolique constituera alors une chaîne de caractères semblables à ce qui a été défini pour les variables temporelles.

3 - Les composants

1° Les noms des registres, mémoires ou bornes

- Ces noms seront exprimés par une chaîne de trois caractères alphanumériques au maximum. Le premier caractère étant obligatoirement alphabétique et différent de T.

2° Les noms de modules

- Ces noms seront exprimés comme les précédents, mais pourront être long de quatre caractères.

4 - Les mots clés

Les mots clés du langage dont l'utilisation et la signification apparaîtront plus loin sont les suivants:

CLOCK DEL END MEM MODULE REG SPEC TIME TML

c) Les opérateurs

1 - Les opérateurs logiques:

Ils n'opèreront que sur les variables logiques et
- le ET sera représenté par le caractère .
- le OU sera représenté par le caractère +
- le ou exclusif sera représenté par le caractère ≠
- le NON sera représenté par le caractère -

2 - Les opérateurs arithmétiques:

Ils n'opèreront que sur les variables ou constantes temporelles et
- l'addition sera représentée par le caractère +
- la soustraction sera représentée par le caractère -
- la multiplication sera représentée par le caractère *
- la division sera représentée par le caractère /

3 - Les opérateurs ∪ (Union) et ∩ (intersection)

Opèreront sur les intervalles de temps

4 - Les opérateurs de transfert :

Il y a trois types de tranfert possibles :
- le tranfert inconditionnel
- le tranfert conditionnel
- le tranfert "3-états" .

Le symbole représentant l'opération de tranfert est le suivant : ←

La condition éventuelle à un tranfert sera notée entre deux barres verticales : | |

Si entre ces barres apparait le symbole "& " cela signifie qu'il y a tranfert "3-états".

I. 3. 3. Règles de description des circuits logiques

a) Structure générale d'une description de module.

La description d'un circuit logique dans le langage "PHPL" aboutit normalement à la production d'un module. Ce module sera décrit sous la structure suivante:

MODULE <nom du module> (<variables d'entrées>)
(<variables de sorties>)

<corps du module>

END

Note: 1) chaque variable d'entrée (de sortie) sera séparée de la précédente par une virgule
2) le mot clé "MODULE" sera séparé du nom du module par un "blanc"

b) Constitution du corps du module

1 - Définition des composants du circuit logique

Le corps du module sera divisé en deux parties. La première correspondra à la mise en place des composants de la description:

1° Déclaration de définition des registres internes du module:

<registres>::= REG <nom de registre>(<MSB>:<LSB>),...,<nom de registre>(<MSB>:<LSB>);

Notes: 1) Il peut n'y avoir qu'un seul registre.
2) MSB = numéro d'ordre du bit le plus élevé
3) LSB = numéro d'ordre du bit le plus bas
4) Entre le mot clé REG et le nom du premier registre il y a un "blanc "
5) Si un registre est d'un bit seulement les ( ) sont omises.

Exemples: 1) REG AC(15:0),MD(15:0);
2) REG FF;

Dans le premier cas nous définissons deux registres AC et MD de 16 bits chacun . Dans le deuxième cas nous définissons un registre FF d'un seul bit.

2° Déclaration de définition des mémoires adressables:

<mémoire> ::= MEM <nom de la mémoire>(<ADL>:<ADH>,<MSB : LSB>);

Notes: 1) Il peut y avoir plusieurs mémoires déclarées simultanément, comme pour les registres, on les sépare par des virgules.
2) ADL = numéro d'adresse inférieure
3) ADH = numéro d'adresse supérieure
4) MSB et LSB : même signification que pour les registres
5) MEM est séparé du nom de la première mémoire par un "blanc"

Exemple:

MEM RAM(0:1023,7:0); soit une mémoire 1K x 8 bits.

3° Déclaration de définition des borniers:

⟨borniers⟩::= TML ⟨nom de bornier⟩(⟨MSB⟩:⟨LSB⟩),...,⟨nom de bornier⟩(⟨MSB⟩:⟨LSB⟩);

Les remarques faites pour les registres s'appliquent dans ce cas-ci également.

Exemple: TML X (15:0);

4° Déclaration de temps (constantes)

Des temps constants ou intervalles de temps exprimés de manière symbolique peuvent être définis par une déclaration TIME :

1) ⟨constante temps⟩::= TIME ⟨nom symbolique de la constante temps⟩=⟨constante temps⟩;

2) ⟨constante temps⟩::= TIME ⟨nom symbolique de la constante temps⟩=⟨expressions⟩;

⟨expression⟩::=⟨constante de temps symbolique ou non⟩⟨opérateur arithmétique⟩ ⟨constante temps⟩...⟨constante temps⟩⟨opérateur arithmétique⟩ ⟨constante temps⟩

3) ⟨intervalle de temps⟩::= TIME ⟨nom de l'intervalle⟩=(⟨DEB⟩,⟨FIN⟩)

⟨DEB⟩::= constante définissant le début de l'intervalle de temps

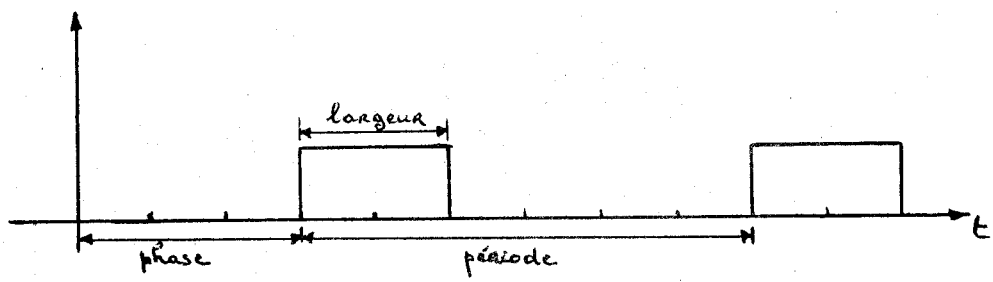
⟨FIN⟩::= constante définissant la fin de l'intervalle de temps

5° Déclaration définissant les variables temporelles

Il est possible de définir un signal d'horloge par une déclaration CLOCK :

⟨signal d'horloge⟩::= CLOCK ⟨nom du signal(temps)⟩(⟨phase⟩,⟨largeur⟩,⟨période⟩);

La figure suivante visualise ce que l'on entend par phase , largeur, période :



Note: phase, largeur et période peuvent être mentionnées par les noms de temps, constantes symboliques préalablement déclarées, ou alors précisées directement.

2 - Description des relations fonctionnelles liant les variables du circuit.

Cette deuxième partie du corps du module définit les relations entre les diverses variables mises en place dans la première partie :

Il y a deux façons de définir ces relations. La première est d'établir soit une relation logique de base (ET, OU, OU exclusif et NON) entre certaines variables soit une simple liaison, ce qui sera décrit au moyen d'une déclaration de tranfert. La seconde manière est de faire appel à un module déjà décrit et d'affecter les diverses entrées et sorties de ce module à des variables du circuit.

1° Définition d'une relation de tranfert:

- 1 - tranfert d'un registre , d'une mémoire ou d'un bornier vers un registre ou une mémoire

a) tranfert inconditionnel

⟨nom de registre ou mémoire destination⟩ ← ⟨nom de registre ou mémoire source⟩;

b) tranfert conditionnel

| ⟨condition⟩ | ⟨nom de registre ou mémoire destination⟩ ← ⟨nom de registre ou mémoire source⟩ ;

Notes : 1) Les noms de registres ou mémoires peuvent être suivis des numéros des lignes pour lesquelles le tranfert est défini.

⟨nom de registre ou mémoire destination⟩ (⟨MSB⟩:⟨LSB⟩) ← ⟨nom de registre ou mémoire source⟩ (⟨MSB⟩:⟨LSB⟩);

2) La condition est représentée par une fonction logique entre variables logiques ou temporelles du circuit :

⟨condition⟩ ::= ⟨nom de variable⟩⟨opérateur logique⟩⟨nom de variable⟩...⟨nom de variable⟩⟨opérateur logique⟩⟨nom de variable⟩

Le nom d'une variable peut être suivi du symbole ↑ (↓) ce qui signifie flanc montant (descendant).

3) Si l'on change d'opérateur il y a lieu d'utiliser les parenthèses selon les règles habituelles.

4) Le coté droit (source) de la relation peut-être l'expression d'une fonction logique entre variables logiques. Un seul type d'opérateur est admis par relation, exception faite de l'opérateur NON :

a) ⟨nom de variable⟩⟨opérateur X⟩⟨opérateur NON⟩⟨nom de variable⟩...

b) ⟨opérateur NON⟩(⟨nom de variable⟩⟨opérateur X⟩⟨nom de variable⟩...
...⟨nom de variable⟩);

c) ⟨nom de variable⟩⟨opérateur X⟩⟨nom de variable⟩...
...⟨opérateur X⟩⟨nom de variable⟩;

- 2 - tranfert d'un registre ou d'une mémoire vers un bornier (BUS)

Les règles sont les mêmes que dans le cas précédent, mais ici il peut apparaître dans l'expression de la condition le symbole '&" qui spécifie un transfert "3- états".

- 3 - introduction d'un délai

Un signal peut être retardé d'un temps déterminé par une déclaration de type:

⟨signal retardé⟩ ← ⟨signal non retardé⟩ DEL ⟨retard⟩;

⟨retard⟩ ::= ⟨temps constant symbolique ou non⟩ | ⟨intervalle de temps⟩

Note : les temps ou intervalles de temps peuvent être donnés par des expressions combinants, au moyens des opérateurs prévus, différents temps préalablement définis.

2° Appel d'un module

L'insertion d'un module préalablement décrit se fera à l'aide du mot SPEC selon la structure suivante:

SPEC ⟨nom du module appelé⟩ ⟨nom donné à la copie actuelle du module⟩ (⟨affectation des entrées du module⟩) (⟨affectation des sorties du module⟩);

⟨affectation des entrées du module⟩ ::= ⟨nom de la première entrée du module⟩ : ⟨nom de la variable du circuit qui lui est affecté⟩, ..., ⟨nom de la dernière entrée du module⟩ : ⟨nom de la variable du circuit qui lui est affecté⟩

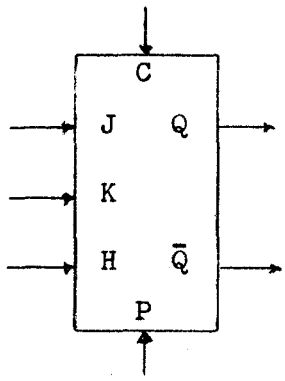
L'affectation de sorties suit la même règle.

Note : Le mot clé SPEC sera séparé du nom du module appelé par un espace d'un caractère.

I. 3. 4. Exemples.

a) Soit a décrire un bistable JK :

Le module JK est représenté à la figure ci-après et sa table de transition est la suivante (basculement sur le flanc descendant du signal horloge H) :



J	K	Q^+
0	0	Q
0	I	0
I	0	I
I	I	$\bar{Q}$

P : mise à 1
C : mise à 0
P et C sont prioritaires sur les autres entrées.

La description en langage "PHPL" est immédiate:

```
MODULE JK(J,K,P,C,H)(Q,NQ);
       REG Q1,NQ1;
       TIME T=10 ns;
       Q ← Q1;
       NQ ← NQ1;
       NQ1 ← -Q1 DELT;
       |P.C.H .J.K| Q1 ← NQ1;
       |P.C.H .J.K̄| Q1 ← 1;
       |P.C.H .J̄.K| Q1 ← 0;
       |P̄.C| Q1 ← 1;
       |P.C̄| Q1 ← 0;
   END
```

b) Afin de donner un exemple de description de module faisant appel à d'autres modules préalablement décrits nous allons synthétiser un compteur par 5.

Ce système décrira un cycle comportant six états, ce qui nous amène à utiliser trois bistables JK.

La table des séquences des états internes est la suivante:

Transitions	y_3	y_2	y_1	y_3^+	y_2^+	y_1^+
1	0	0	0	0	0	1
2	0	0	1	0	1	0
3	0	1	0	0	1	1
4	0	1	1	1	0	0
5	1	0	0	1	0	1
6	1	0	1	0	0	0

Le système étant simple il peut-être synthétisé a l'aide de bistables de type T. Comme nous ne disposons que de module JK, nous placerons J et K à 1 et T correspondra à l'entrée horloge.

La table de vérité est la suivante:

H	y_3	y_2	y_1	y_3^+	y_2^+	y_1^+	T_3	T_2	T_1
↓	0	0	0	0	0	1	0	0	1
↓	0	0	1	0	1	0	0	1	1
↓	0	1	0	0	1	1	0	0	1
↓	0	1	1	1	0	0	1	1	1
↓	1	0	0	1	0	1	0	0	1
↓	1	0	1	0	0	0	1	0	1

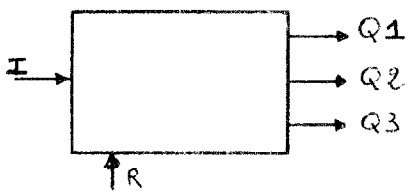
La minimisation des fonctions d'excitation donne les résultats suivant:

$$T_1 = H\downarrow$$

$$T_2 = \bar{y}_3 \cdot y_1 \cdot H\downarrow$$

$$T_3 = y_1 \cdot (y_2 \oplus y_3) \cdot H\downarrow$$

Le module compteur par 5 est schématisé à la figure ci-dessous, nous l'appelerons CPT5 :



Sa description en langage PHPL sera alors :

```
MODULE CPT5(I,R)(Q1,Q2,Q3);
       TML I2,I3,Y,Y1;
       I2 ← Y.Q1.I;
       Y1 ← Q2≠ Q3;
       I3 ← Q1.Y1.I;
       SPEC JK=FF1(J:1,K:1,P:1,C:R,H:I)(Q:Q1,NQ);
       SPEC JK=FF2(J:1,K:1,P:1,C:R,H:I2)(Q:Q2,NQ);
       SPEC JK=FF3(J:1,K:1,P:1,C:R,H:I3)(Q:Q3,NQ:Y);
   END
```

Remarquons que nous pouvons ainsi créer différents modules compteurs, additionneurs et autres. Puis, à partir de ceux-ci, en créer de nouveaux et arriver de cette façon à des systèmes fort complexes.

I. 4. LE PROGRAMME EDITEUR.

I. 4. 1. Introduction.

Le système sur lequel est basé cette étude est le GENIUS à microprocesseur 2650 de signétics. On trouvera à l'annexe 1-1 les caractéristiques principales de ce système.

Ce programme réalise la procédure que nous avons prévue en début de ce chapitre en suivant l'ordinogramme qui est repris à l'annexe 1-2.

Pour aider la lecture de cet ordinogramme nous devons donner quelques précisions sur la gestion de l'écran à partir du clavier.

Le clavier interrompt le microprocesseur qui vient le lire. Comme cette interruption est exécutée en adressage indirect, il suffit d'écrire par programme, dans une place mémoire pré-déterminée, l'adresse de départ de la sous-routine d'interruption que l'on désire effectuer. Nous appelerons le fait d'écrire cette adresse de départ dans la place mémoire voulue: charger le vecteur d'interruption.

La sous-routine d'interruption ira normalement lire le code déposé sur la porte correspondant au clavier et après l'avoir idéntifié effectuera la ou les tâches prévues. Eventuellement le caractère correspondant à ce code, ou un autre, pourra être écrit sur l'écran à l'aide de la sous-routine moniteur appropriée.

Notons encore que l'interface TRC vient lire le contenu d'une mémoire dans laquelle le microprocesseur, lui, vient écrire. Chaque adresse de cette mémoire correspond à une position sur l'écran. Il est donc aisément possible, connaissant le nombre de caractères sur une ligne, et le nombre de lignes sur l'écran, de passer des coordonnées x, y d'un point à l'adresse mémoire correspondante (A condition, bien sur, de connaitre l'adresse de départ de la mémoire en question et de savoir à quel point de l'écran elle correspond: le coin supérieur gauche dans le cas présent.).

Enfin le moniteur du GENIUS permet de n'écrire que dans une zone déterminée de l'écran que l'on fixe en déterminant le cadre de la façon suivante:

- bord supérieur
- bord inférieur
- bord gauche
- largeur

Nous avons, en utilisant cette facilité, partagé l'écran en 4 zones:

1 - la zone "commentaire": les deux demi-lignes inférieures gauche

2 - la zone "Menu":le $\pm$ 1/4 droit de l'écran (12 derniers caractères)

3 - la zone "Visualisation de fonctions et modules": la 1/2 droite du reste de l'écran (- 2 lignes inférieures)

4 - la zone "d'édition des descriptions": la 1/2 gauche du reste de l'écran (- 2 lignes inférieures).

I. 4. 2. Organisation des données:

Pour cette partie du travail nous avons créé quatre tables:

1) TE la table d'entrée:

Elle contiendra le nom de la description en élaboration; ainsi que les noms des entrées et sorties.

Structure: a) PDE : pointeur de début de table (fixe)
b) PFE : pointeur de fin de table (variable)
c) Tous les noms seront codés en ASCII, un octet par caractère
d) Le nom du circuit aura au maximum 4 caractères, les autres noms auront au maximum 3 caractères.

e) Le nom du circuit sera séparé du 1° nom de ligne d'entrée par un octet ayant la valeur H'FF'.

f) Chaque nom d'entrée (sortie) sera séparé du suivant par la valeur H'11'.

2) TFM la table des fonctions et modules:

Contiendra les noms des fonctions ou modules retenus par l'utilisateur.

Structure: semblable à la table TE.
Chaque fonction ou module sera représenté par un numéro (2 caractères, un "blanc" , nom en 8 caractères).

3) TD la table des descriptions des fonctions et modules retenus en langage PHPL:

Structure: suivant règles du langage PHPL (chaque caractère étant codé en ASCII sur un octet).Chaque ligne comprend 20 caractères au maximum.

- PDD : pointeur de début de table (fixe)
- PFD : pointeur de fin de table (variable)

4) TED la table contenant la description éditée:

Table remplie lors de l'édition, c.à.d pendant la sous-routine SRI suivant dénomination reprise dans l'ordinogramme éditeur.

Structure: identique à TD.

- PD : pointeur de début de table
  PD = PFE + 1
- PF : pointeur de fin de table.

I. 5. CONCLUSION.

Le problème posé dans le cahier des charges a, au stade actuel du travail, trouvé une solution possible donnée sous la forme d'une description fonctionnelle contenue dans la table TED.

Cette solution ne pourra néanmoins devenir définitive qu'après avoir été évaluée.

C'est pourquoi dans la phase suivante du travail, objet du chapitre II, nous allons simuler le circuit logique correspondant à la description fonctionnelle et voir comment il répond à diverses sollicitations d'entrée .

C H A P I T R E II

S I M U L A T E U R.

II. 1. INTRODUCTION.

Nous allons dans ce chapitre chercher à vérifier si la description fonctionnelle produite précédement répond au cahier des charges.

Notons que si cette vérification s'avère satisfaisante, nous transposerons la dite description fonctionnelle en un schéma matériel que nous devrons également tester.

Nous chercherons donc à créer un simulateur pouvant s'adapter aux deux niveaux de fonctionnement.

La simulation se fera essentiellement en deux phases:

1 - la modélisation

2 - l'évaluation

La modélisation consiste à traduire la description fonctionnelle (ou le schéma matériel) , en machine, sous une forme directement exploitable par un programme procédant à l'évaluation.

L'évaluation consiste à exécuter un programme, se servant du modèle créé à la phase précédente, afin de rendre compte du comportement des sorties du circuit logique décrit, pour une séquence de signaux donnée aux entrées.

Il y aura lieu de bien envisager la manière d'introduire les séquences des signaux d'entrée devant servir à l'évaluation. C'est là en effet qu'il y aura lieu, pour ce chapitre, de rendre le système interactif.

Lors de la modélisation nous pourrons vérifier également si aucune entrée ou sortie n'a été oubliée ou "connectée" de façon incorrecte dans la description fonctionnelle.

II. 2. LA MODELISATION.

II. 2. 1. Modélisation du temps.

Le temps sera quantifié en "unités de temps". La valeur assignée à ces unités est relative. Chaque temps intervenant dans la simulation sera exprimé par un nombre entier de fois cette unité.

La simulation pourra s'effectuer ou non en synchronisme avec un signal horloge.

II. 2. 2. Modélisation des valeurs d'un signal.

Il est bien évident qu'en logique binaire, un signal ne peut avoir que la valeur 0 ou 1. Il est vrai aussi toutefois, qu'en pratique un signal devant passer de 0 à 1, par exemple, aura pendant un certain

temps une valeur qui ne sera ni 0 ni 1 : nous dirons que cette valeur est indéterminée et nous l'appellerons X.

Nous allons montrer qu'en modélisant les valeurs d'un signal de cette manière il est possible de mettre assez facilement en évidence dans un circuit des phénomènes de course et de détecter les endroits ou pourraient survenir des "aiguilles".

Prenons la cas d'une porte ET à 2 entrées, A et B, la sortie étant appellée S. Si nous envisageons les 3 valeurs, 0, X et 1 pour chaque entrée ou sortie, nous aurons la table de vérité suivante:

A	B	S
0	0	0
0	X	0
0	1	0
X	0	0
X	X	X
X	1	X
1	0	0
1	X	X
1	1	1

Examinons à présent l'évolution de la sortie lorsque l'entrée A passe de 0 à 1 et l'entrée B de 1 à 0 :

A	B	S
0	1	0
X	X	X
1	0	0

Nous voyons donc la sortie passer par un état indéterminé X, alors que normalement, en logique binaire pure, la transition envisagée aux entrées ne devrait provoquer aucune variation de la sortie.

La modélisation "3 valeurs" peut donc servir à détecter les phénomènes de courses possibles et l'apparition "d'aiguilles", ce qui surviendrait effectivement dans le cas présent si une des deux entrées commutait plus rapidement que l'autre.

Cette valeur X peut aussi servir pour représenter les valeurs d'entrées non spécifiées en début de simulation et en mesurer l'influence.

On peut aussi, grace à la valeur X, s'assurer rapidement de l'efficacité de certaines commandes comme la remise à zéro; il suffit de mettre à X les entrées et de vérifier si cette valeur s'est propagée jusqu'à la sortie d'une fonction devant être remise à O.

Plus de 3 valeurs peuvent être envisagées, ainsi M. TOKORA et autres (51) utilisent une représentation avec " 4 valeurs " :
O, D, U, 1.

Avec D correspondant à la valeur indéterminée proche de O et U correspondant à la valeur indéterminée proche de 1.

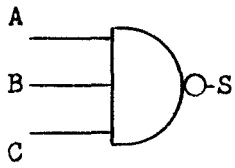
II. 2. 3. Modélisation d'une fonction.

Chaque constituant, fonction plus ou moins complexe, du circuit logique à simuler devra avoir été préalablement modélisé et mémorisé dans une banque de données.

La modélisation consistera à produire pour chaque fonction un sous-programme réalisant au niveau du processeur ce qu'elle réalise dans le circuit logique.

Nous allons donner, à titre d'exemple, la modélisation possible d'une porte NAND à trois entrées en langage assembleur 2650 :

Soit




```
1  LODA, RO  A
2  ANDA, RO  B
3  ANDA, RO  C
4  LODI, R1  H'FF'
5  EORZ, R1
6  STRA, RO  S
7  RETC, UN
```

Chaque fois que l'on voudra simuler une fonction NAND à 3 entrées on pourra appeller ce sous-programme. Bien sur il faudra à chaque fois spécifier l'adresse des signaux A, B, C et S.

II. 2. 4. Modélisation parallèle.

Nous venons de parler des adresses des signaux A, B, C et S C'est qu'en fait, normalement, dans un programme de simulation, un mot entier du processeur est utilisé pour un signal et il ne représente qu'une seule valeur instantanée de ce signal.

Or comme les instructions du processeur manipulent chaque bit du mot de donnée de manière indépendante, il est possible de traiter simultanément toute une séquence.

Il suffit pour cela de considérer que le bit 0 corresponde à un temps t0, le bit 1 a un temps t1 et ainsi de suite jusqu'au temps t7

représenté par le bit 7 dans le cas de mots de 8 bits.

Le temps de traitement peut ainsi être notablement réduit.

II. 2. 5. Modélisation d'un circuit logique.

La modélisation d'un circuit logique se fera, comme nous l'avons déjà dit, en traduisant la description fonctionnelle sous une forme exploitable par le programme procédant à l'évaluation. En fait il y a deux grandes options possibles, soit réaliser un programme faisant appel, dans l'ordre voulu, aux sous-programmes modélisant les différentes fonctions du circuit, soit en créant une structure de données représentant le réseau.

La première méthode correspond à une technique de compilation, la seconde à une technique se basant sur ce que nous appelerons des "Tables-guides".

a) Modélisation par compilation (31)

Dans cette technique on construit donc directement le programme qu'il suffira d'exécuter pour procéder à l'évaluation. Ce programme devra faire appel, dans l'ordre voulu, aux sous-programmes représentant les fonctions du circuit. Cet ordre correspond bien sur à celui suivant lequel les fonctions logiques sont activées dans le circuit.
Cet ordre devra être déterminé avant de générer le programme propement dit.

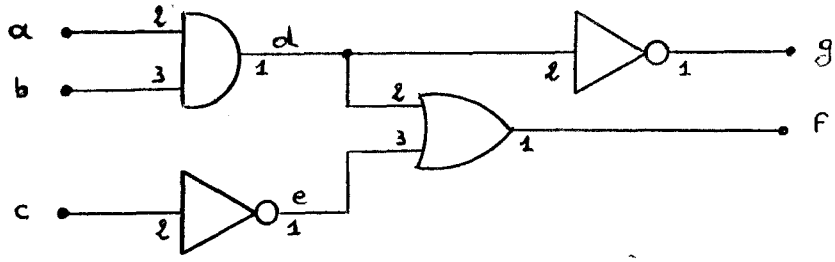
b) Technique des "tables-guides". (51)

La description fonctionnelle n'est plus ici traduite en un programme directement exécutable, mais bien en une structure de données représentant le circuit logique. Le circuit est décrit à l'aide de tables

ou l'on défini les caractéristiques des fonctions, l'origine de chacune de leurs entrées et la destination de chacune de leurs sorties.

Ces tables sont ensuite utilisées par un programme que l'on appelle interpréteur, et qui exécutera dans l'ordre voulu les sous-routines représentant les fonctions du circuit.

Nous allons préciser quelque peu cette technique par un exemple simple: Soit le circuit suivant :



a, b, c, d, e, f, et g représentent les signaux et les nombres 1, 2, 3 représentent les numéros associés à chaque entrée ou sortie d'une fonction (broches).

Nous allons créer trois tables:

- La table de base
- La table source
- La table destination

La table de base: - spécifie par un numéro code le type de fonction délivrant chacun des signaux.
Par exemple : 1 - entrée du circuit
2 - porte ET
3 - porte Ou
4 - inverseur
etc

- précise, par un pointeur, les adresses de la table suivante (table source) concernées par la fonction délivrant le signal examiné.

La table source : - donne pour chaque entrée de la fonction concernée le nom et le numéro de sortie de sa fonction source.

- précise, par un pointeur, pour chaque sortie, les adresses de la table suivante qui les concerne.

La table destination:- donne les noms et numéros d'entrée des fonctions destinations de chaque sortie.

Notes: - Le nom d'une fonction dans les 2° et 3° tables correspond en fait à son numéro d'ordre dans la table de base.
- Un zéro dans une table indique qu'il n'y a pas d'information correspondante à donner pour le signal concerné.
- Un * indique une sortie non interconnectée.

Pour le circuit précédent nous aurons dès lors les tables suivantes

1- **Table de base (TB)**

n°	type de fonction	signaux	pointeur TS
1	1	a	1,1
2	1	b	2,2
3	1	c	3,3
4	2	d	4,6
5	4	e	7,8
6	3	f	9,11
7	4	g	12,13

2- Table source (TS)

n°	élément source	pointeur TD
1	0	1,1
2	0	2,2
3	0	3,3
4	0	4,5
5	1	0
6	2	0
7	0	6,6
8	3	0
9	0	0
10	4,1	0
11	5,1	0
12	0	0
13	4,1	0

3- Table destination (TD)

n°	élément destination
1	4,2
2	4,3
3	5,2
4	7,2
5	6,2
6	6,3

Nous voyons sans peine que cette technique permet, au contraire de la technique de compilation, des modifications rapides et simples du circuit.

II. 3. L' EVALUATION.

Nous disposons actuellement de la structure du circuit sous une forme exploitable et d'un ensemble de sous programmes pouvant simuler chacune des fonctions. Il nous reste à réaliser les phases suivantes:

- Etablir les conditions initiales.
- Introduire les séquences de signaux d'entrées dont on veut évaluer l'effet sur le circuit.
- Evaluer le circuit en réalisant les opérations programmées
- Spécifier la forme suivant laquelle on fournit les résultats.

II. 3. 1. Introduction des données d'entrée

Les conditions initiales pourront être fournies, en général, en même temps que les séquences des signaux d'entrées. Ces séquences pourront être décrites de diverses manières. Par exemple, comme des formes d'ondes, ou comme des tables donnant les valeurs des différents signaux à différents instants.

L'essentiel étant de rendre l'introduction de ces séquences la plus aisée possible et de pouvoir réutiliser une même séquence plusieurs fois sans devoir la réintroduire à chaque fois. Des langages spécifiques peuvent être développés à cette fin.

II. 3. 2. Algorithmes de simulation

Ces algorithmes dépendront directement des choix fait au niveau de la modélisation.

Ces choix pourront porter sur les valeurs possibles par signal:

soit deux valeurs (0,1)
soit trois valeurs (0, X, 1)

Ils pourront aussi porter sur le mode:

compilatif ou interprétatif par tables-guides.

Dans ce qui suit nous donnerons quelques exemples (44)

a) Algorithme basé sur une modélisation "3 valeurs"

Cet algorithme se déroule en deux passes: la passe "X" et la passe "valeur".

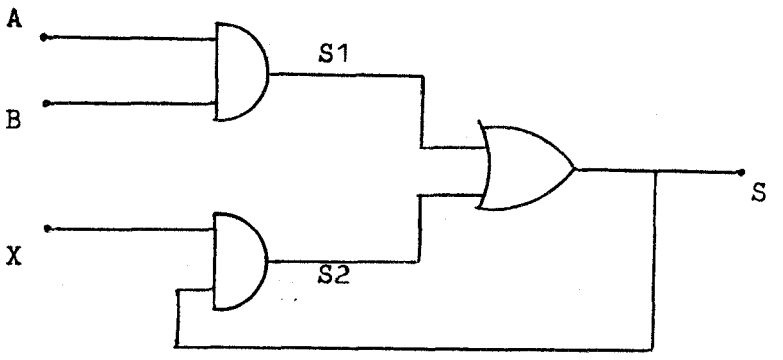
Durant la passe "X" les entrées du circuit changeant d'état sont assignées à la valeur intermédiaire X. Ces valeurs sont propagées selon les tables de vérité "3 valeurs".

Durant la passe "valeur" les entrées du circuit sont amenées à leur nouvelle valeur 0 ou 1. Ces nouvelles valeurs sont propagées à leur tour, ce qui termine la simulation de la phase en cours.

Une valeur "X" persistant de la passe "X" à la passe "valeur" identifie un phénomène de course, puisque la logique a atteint un état stable qui doit être soit 1, soit 0.

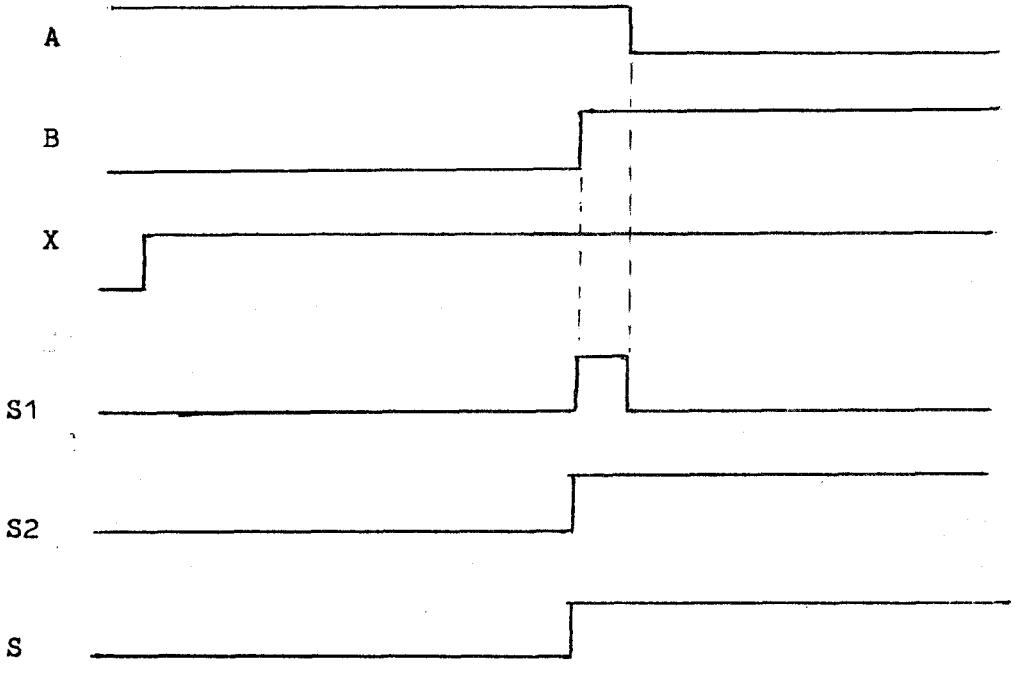
Pour illustrer cette technique nous allons reprendre une porte ET à deux entrées, ou nous avons déjà montré le phénomène de course qui peut survenir lorsque les valeurs des deux entrées s'inversent simultanément.

La porte ET sera suivie d'un circuit simple permettant de mémoriser le basculement de la porte, une ligne X permettra l'initialisation du système. Nous aurons donc le circuit suivant:



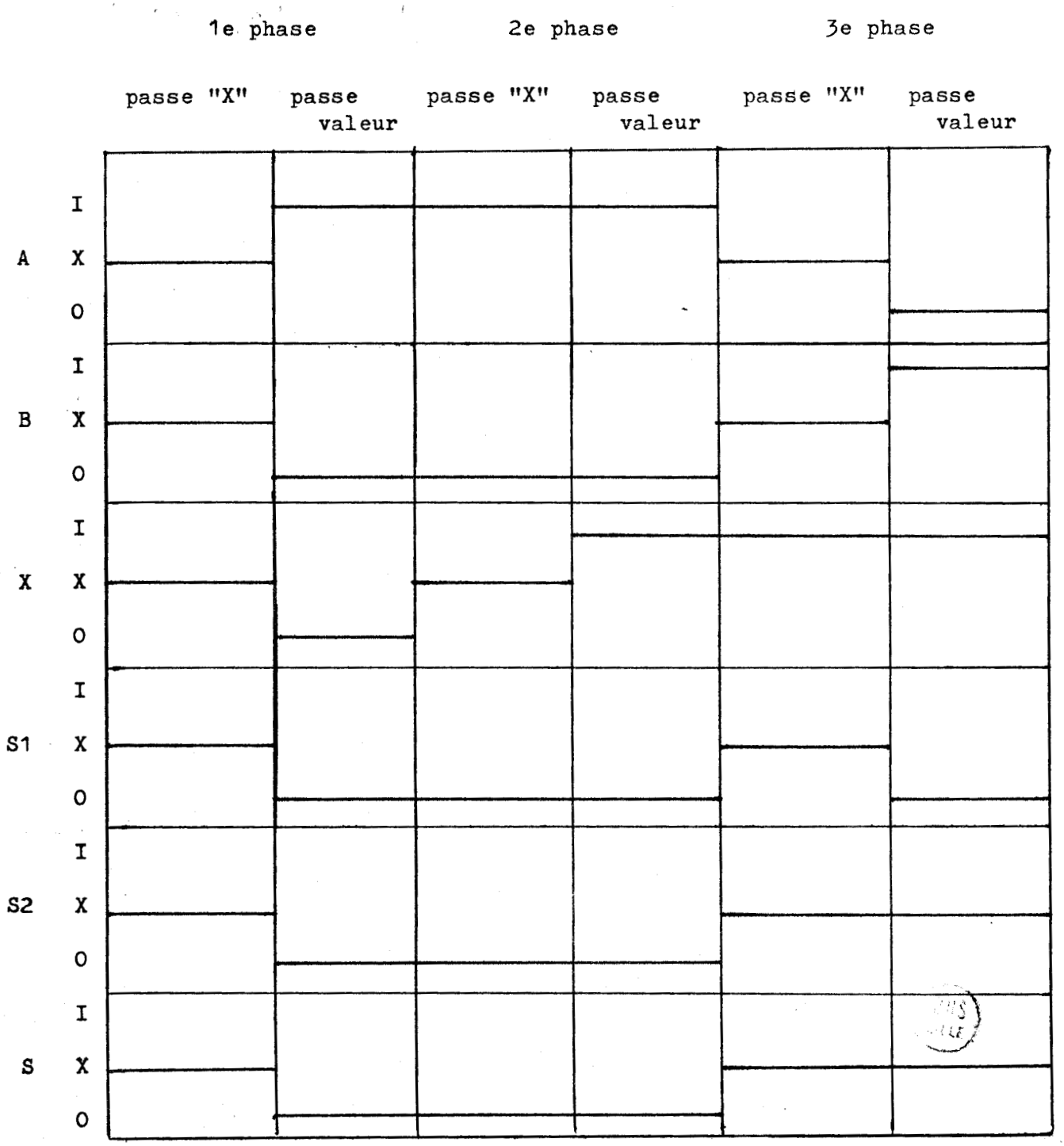
Supposons au départ A = 1, B = 0 . Pour déterminer l'état de S nous mettons l'entrée X à 0 un court instant, après quoi nous la maintiendrons à 1.

Le diagramme suivant montre bien que S passe à 1 et s'y maintient lorsque les basculements de A vers 0 et B vers 1 ne se font pas simultanément.



Voyons maintenant ce que donne l'application à ce circuit de l'algorithme "3 valeurs" en deux passes:

1° phase : Reset A = 1, B = 0, X = 0
2° phase : Début A = 1, B = 0, X = 1
3° phase : Commutation A = 0, B = 1, X = 1



Donc en fin de simulation S et S2 sont toujours dans l'état "X" ce qui met en évidence le problème de course inhérent à la structure de ce circuit.

b) Algorithme basé sur une modélisation par compilation.

Dans le cas présent l'algorithme doit d'abord ordonner les fonctions de manière à s'assurer que les valeurs de leurs sorties soient calculées avant d'être utilisées dans la suite du traitement.

Les chaînes de parenthèses utilisées dans les expressions booléennes des langages de description permettent de définir simplement le niveau de chaque fonction. Un compteur de niveau est incrémenté pour chaque parenthèse gauche et décrémenté pour chaque parenthèse droite.

Exemple:

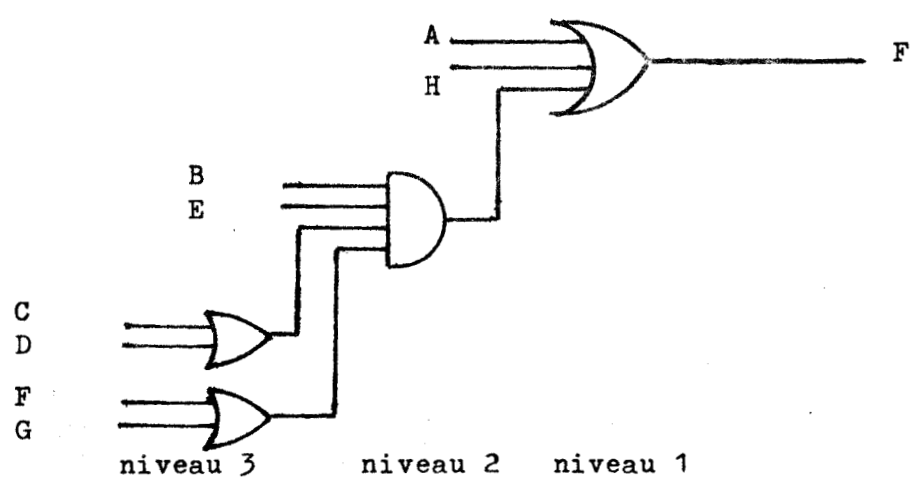
Soit F = (A + (B. (C + D) .E. (F + G)) + H)

niveau 1 : (A + + H)

niveau 2 : (B. .E.)

niveau 3 : (C + D) (F + G)

Or si nous représentons F par un schéma logique nous aurons:



Nous voyons donc qu'il faut évaluer les fonctions du niveau le plus élevé, puis celles du niveau immédiatement inférieur et ainsi de suite jusqu'au niveau 1.

c) Algorithme correspondant à une simulation synchronisée sur "l'événement suivant".

En se synchronisant sur "l'événement suivant" plutôt que sur le "prochain temps horloge", on ne devra évaluer que les fonctions dont l'état de sortie risque de changer (parce qu'une de leurs entrées au moins vient juste de changer).

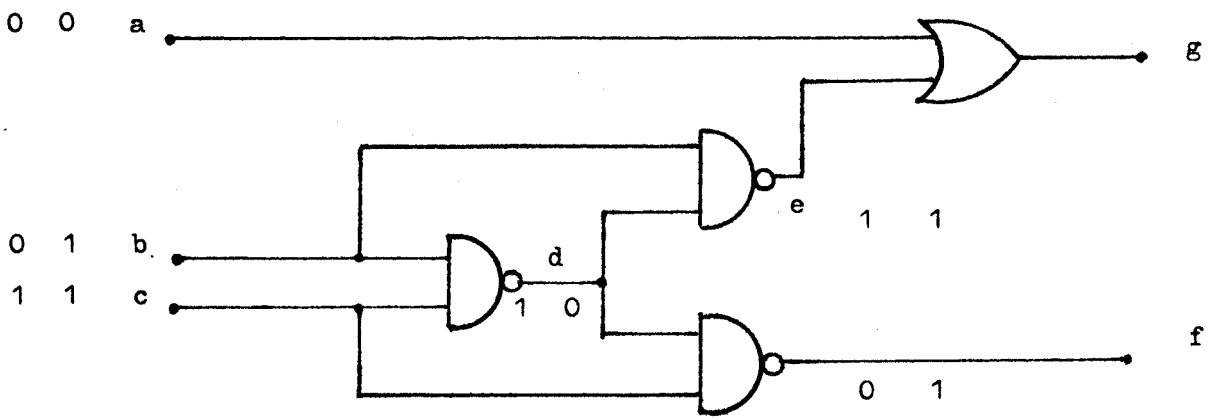
Cette procédure utilisera deux piles du type LIFO pour représenter les événements actuels et futurs.

Toutes les fonctions dont l'état peut changer à un moment donné (où survient un événement) sont placées dans une première pile, S0, appellée pile "d'activité en cours". Au temps unitaire suivant, chaque fonction dans S0 en est retirée et examinée. Si son état a changé, les fonctions vers lesquelles ses signaux de sorties sont propagés, soit l'ensemble de ses fonctions destination, sont placées dans une seconde pile, S1, appellée pile "d'activités futures". S1 contient donc tous les candidats au changement d'état. Toutefois on n'évalue pas les fonctions dans S1 avant que S0 ne soit vide.

Lorsque tous les éléments dans la pile S0 ont été examinés, les rôles des deux piles sont inversés, S1, devient la pile "d'activités en cours" et S0, la pile "d'activités futures", ce qui permet d'évaluer les éléments candidats se trouvant dans S1.

Le processus est répété, vidant une pile et remplissant l'autre jusqu'à ce qu'elles soient toutes les deux vides.

Prenons comme exemple le circuit suivant:



Empiler : constitue l'action de mettre un élément dans la pile

Désempiler : constitue l'action de retirer un élément de la pile et de l'évaluer.

Actions	Arguments				Commentaires		
	fonction (signal)	Pile	Contenu de S_0	S_1	Pile "d'activité en cours"	Nouvelle valeur	Ancienne valeur
empiler	a	S_0	a	0	-	-	-
empiler	b	S_0	a,b	0	-	-	-
empiler	c	S_0	a,b,c	0	-	-	-
désempiler	c	S_0	a,b	0	S_0	1	1
désempiler	b	S_0	a	0	S_0	1	0
empiler	e	S_1	a	e	S_0	-	-
empiler	d	S_1	a	e,d	S_0	-	-
désempiler	a	S_0	0	e,d	S_0	0	0
désempiler	d	S_1	0	e	S_1	0	1
empiler	e	S_0	e	e	S_1	-	-
empiler	f	S_0	e,f	e	S_1	-	-
désempiler	e	S_1	e,f	0	S_1	1	1
désempiler	f	S_0	e	0	S_0	1	0
désempiler	e	S_0	0	0	S_0	1	1

d) Variante du cas précédent tenant compte des délais introduits par chaque fonction.

Dans la technique que nous venons de voir tous les événements futurs sont modélisés comme s'ils étaient exécutés et réalisés au même moment. Dans la technique que nous allons exposer la simulation pourra tenir compte de plusieurs événements qui se déroulent différemment dans le temps mais qui seront traités simultanément.

Chacun des moments auquels les événements seront réalisés sera un multiple d'une unité de temps quantifié.

Si les activités futures sont représentées en utilisant plus qu'une pile de type LIFO, les événements pourront prendre place à autant d'instants différents qu'il y a de piles.

Le temps sera quantifié selon un intervalle Δt. Ainsi, chacun des instants $t_1, t_2, \ldots\ldots, t_i, t_{i+1}, \ldots\ldots, t_n$, auxquels pourront survenir les "activités futures", seront défini par la règle:

$$t_{i+1} = t_i + \Delta t$$

- Pratiquement Δt sera le plus grand commun diviseur des délais individuels attachés à chaque fonction.

- l'ensemble du temps couvert par les intervalles Δt sera

  $(d\ max + 1).\Delta t$, où d max est le délai le plus grand devant être considéré.

- Tout changement aux entrées de la fonction x au temps t_i est évalué au temps $t = t_i + d_x$ où d_x est un entier multiple de Δt.

Le temps t, spécifie la pile S_t dans laquelle les "fonctions destinations" de x sont placées.

- Un pointeur P pointe la pile "d'activité en cours", S_i, progressant vers la pile suivante correspondant à l'instant auquel l'événement suivant survient.

- Les intervalles Δt forment une boucle circulaire de temps t_1, t_2 .....t, t + 1, ...t_n correspondant à une boucle circulaire de piles pointées successivement par P :

$$S_1, S_2, \ldots\ldots, S_t, S_{t+1}, \ldots., S_n, S_1, S_2, \ldots.$$

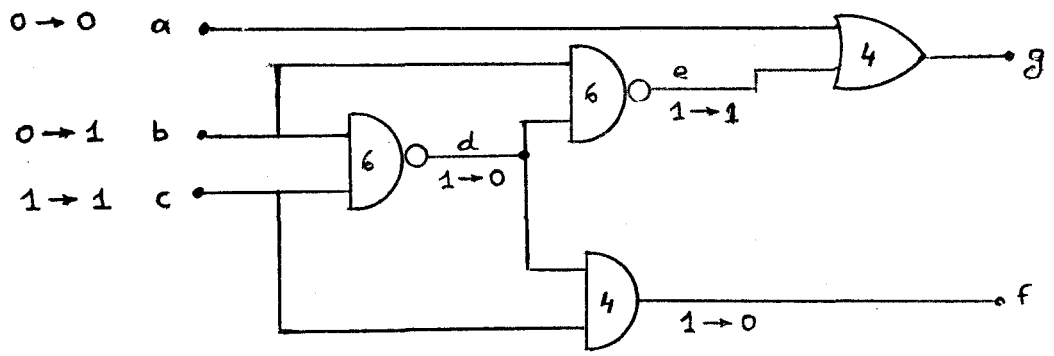
- Chaque fonction contenue dans la pile actuelle "d'activité en cours" en est retirée et évaluée. Chaque fonction destination dont les signaux d'entrées ont changés est placée dans la pile appropriée.

- Le pointeur avance à la pile suivante de la boucle quand la pile "d'activité en cours" est vidée.

- La procédure s'arrête lorsque toutes les piles sont vides.

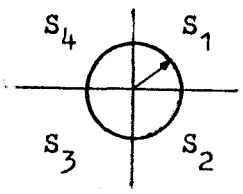
Nous allons reprendre le circuit suivant à titre d'exemple:

Nous avons pour chaque fonction noté le délai introduit:



Le p g c d de 6 et 4 étant 2, nous aurons $\Delta t = 2$
d'ou n, le nombre de piles sera:

$$\frac{d\ max}{\Delta t} + 1 = \frac{6}{2} + 1 = 4$$



Actions	Piles				temps	nouvelles valeurs	anciennes valeurs
	S_1	S_2	S_3	S_4			
1 empiler (a,S_1)	a				0	-	-
2 empiler (b,S_1)	a,b				0	-	-
3 empiler (c,S_1)	a,b,c				0	-	-
4 désempiler (c,S_1)	a,b				1	1	1
5 désempiler (b,S_1)	a				1	1	0
6 empiler (e,S_4)	a			e	1	-	-
7 empiler (d,S_4)	a			e,d	1	-	-
8 désempiler (a,S_1)	0			e,d	1	0	0
9 désempiler (d,S_4)	0			e	4	0	1
10 empiler (e,S_3)	0		e	e	4	-	-
11 empiler (f,S_2)	0	f	e	e	4	-	-
12 désempiler (e,S_4)	0	f	e	0	4	0	1
13 empiler (g,S_2)	0	f,g	e	0	4	-	-
14 désempiler (g,S_2)	0	f	e	0	6	0	1
15 désempiler (f,S_2)	0	0	e	0	7	0	1
16 désempiler (e,S_3)	0	0	0	0	8	1	0
17 empiler (g,S_1)	g	0	0	0	9	-	-
18 désempiler (g,S_1)	0	0	0	0	11	1	0

Pour une partie de circuit et une séquence de signaux d'entrées identiques, nous voyons apparaitre un phénomène de course non décelé par la méthode précédente.

II. 4. CHOIX D'UNE TECHNIQUE DE SIMULATION.

L'étude que nous menons ayant pour but d'implanter un système interactif, il nous semble évident que nous choisirons une technique basée sur les "tables- guides" plutôt qu'une technique de compilation. Ces"tables-guides" nous aiderons également pour la vérification des erreurs ou ommissions au niveau de la description fonctionnelle (entrées ou sorties oubliées ou incorrectement connectées). De plus elles nous servirons également pour la transposition "matérielle".

Notre intention étant de pouvoir tenir compte, dans la simulation, des délais introduits par les différentes fonctions nous opterons pour une simulation du type que nous avons vu en dernier lieu.

Par contre nous avons vu que cette technique pouvait, en tenant compte des délais introduits par les fonctions, faire ressortir les phénomènes de courses. Nous nous contenterons donc d'une représentation "deux valeurs".

II. 5. IMPLANTATION D'UN SIMULATEUR.

II. 5. 1. La modélisation

Pour modéliser le circuit à simuler nous allons devoir créer trois tables-guides: la table de base (TB), la table source (TS), et la table de destination (TD). Nous nous servirons pour cela de la

table contenant la description du circuit en langage PHPL. Cette table nous l'avons appelée dans le chapitre précédent la table TED (Table contenant la Description Editée). Nous nous servirons également d'une autre table, la table de codage des fonctions et modules (TCM), qui traduira le nom de chaque fonction ou module en un code d'un octet.

a) Création de la table de base (TB)

La table de base contiendra autant d'objets qu'il y a de signaux dans le circuit à modéliser. Chaque objet sera représenté dans la table par le contenu de 7 octets successifs:

- les 3 premiers contiendront le nom du signal en code ASC II
- le 4è contiendra le "code fonction"
- les deux suivants contiendront l'adresse de la zone de la table TS correspondant à cet objet.
- le dernier contiendra le nombre de places de la table TS qui sont affectées à cet objet.

Trois pointeurs permettront de se servir de cette table:

- PDB, le pointeur de début de table (fixe)
- PB , le pointeur de remplissage de la table (compteur).
- PFB, pointeur de fin de table (variable).

L'objet n° X de TB sera pointé par

$$PB_X = PDB + 7\ X$$

Pour créer la table TB, il faudra effectuer trois passes:

- La première passe consiste à lire dans la table TED et dans l'ordre:
  - les noms des entrées du circuit
  - les noms des REG
  - les noms des TML
  - les noms des sorties du circuit.

Ces noms seront transcrit dans les 3 premiers octets des objets successifs de TB. Pour les objets correspondant aux noms des entrées du circuit on pourra remplir simultanément les autres octets (4 et 7)

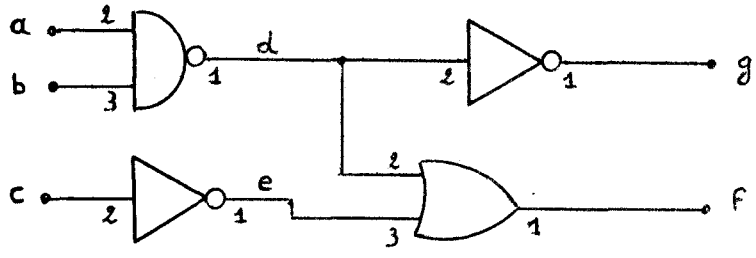
- Lors de la deuxième passe on lit la table TED en détectant les caractères "◄— " et "SPEC", ce qui permet avec l'aide de la table TCM de remplir pour chaque objet les 4e et 7e octets.

- Enfin la troisième passe permet de remplir les octets 5 et 6.

L'ordinogramme correspondant à la création de la table de base se trouve à l'annexe 2.1

Exemple:

Reprenons le circuit utilisé précedemment, soit:



Sa description en langage PHPL sera:

```
MODULE  EX ( a, b, c ) ( g, f ) ;
TML  d, e;
d ← a. b;
e ← - c;
f ← d + e;
g ← - d;
END
```

La table TED correspondante sera:

```
PD
↓
MODULE bl EX ( a, b, c ) ( g, f ) ; TML bl d, e;
d ← a; e ← - c; f ← d + e; g ← - d; END
                                    ↑
                                   PFD
```

La table TB correspondante sera:

PREMIERE PASSE

	a	b	c	d	e	f	g
CODE	41	42	43	44	45	46	47
ASCII	00	00	00	00	00	00	00
	00	00	00	00	00	00	00
FONCTION	01	01	01				
ADRESSE							
NOMBRE	01	01	01				

DEUXIEME PASSE

	a	b	c	d	e	f	g
CODE	41	42	43	44	45	46	47
ASCII	00	00	00	00	00	00	00
	00	00	00	00	00	00	00
FONCTION	01	01	01	02	08	03	08
ADRESSE							
NOMBRE	01	01	01	03	02	03	02

TROISIEME PASSE

	a	b	c	d	e	f	g
CODE	41	42	43	44	45	46	47
ASCII	00	00	00	00	00	00	00
	00	00	00	00	00	00	00
FONCTION	01	01	01	02	08	03	08
ADRESSE	00	00	00	00	00	00	00
	01	05	09	0D	19	21	2D
NOMBRE	01	01	01	03	02	03	02

b) Création de la table source (TS)

Le nombre d'objets de la table source est déterminé lors de la création de la table de base. Chaque objet sera représenté dans la table par 4 octets successifs.

- le 1er octet contiendra le n° de l'objet de la table TB qui fournit le signal source pour l'objet considéré de TS
- les 2è et 3è octets contiendront l'adresse de la zone de la table TD (table destination) correspondant à cet objet de TS.
- le dernier octet contiendra le nombre de places de la table TD affectés à cet objet.

Trois pointeurs permettront de se servir de cette table:

- PDS, le pointeur de début de table
- PFS, le pointeur de fin de table
- PS , le pointeur courant

L'objet n° X de TS sera pointé par:

$$PS_X = PDS + 4\ X$$

Pour créer cette table il faudra effectuer trois passes:

- La première, dont l'ordinogramme est donné ci-après, permettra d'inscrire dans TS le signal source correspondant à chacun de ses objets.

- La deuxième passe, qui ne pourra se faire qu'après création de la table destination, permettra de compléter le 4è octet de TS.

- La troisième sera identique à la troisième passe de création de TB, sans toutefois tenir compte des modules.

c) Création de la table destination (TD)

Chaque objet de la table sera représenté par le contenu de deux octets:

- le premier contiendra le n° de l'objet de la table de base correspondant à la fonction destination.
- le deuxième donnera le n° de la broche de cette fonction qui constitue la destination du signal.

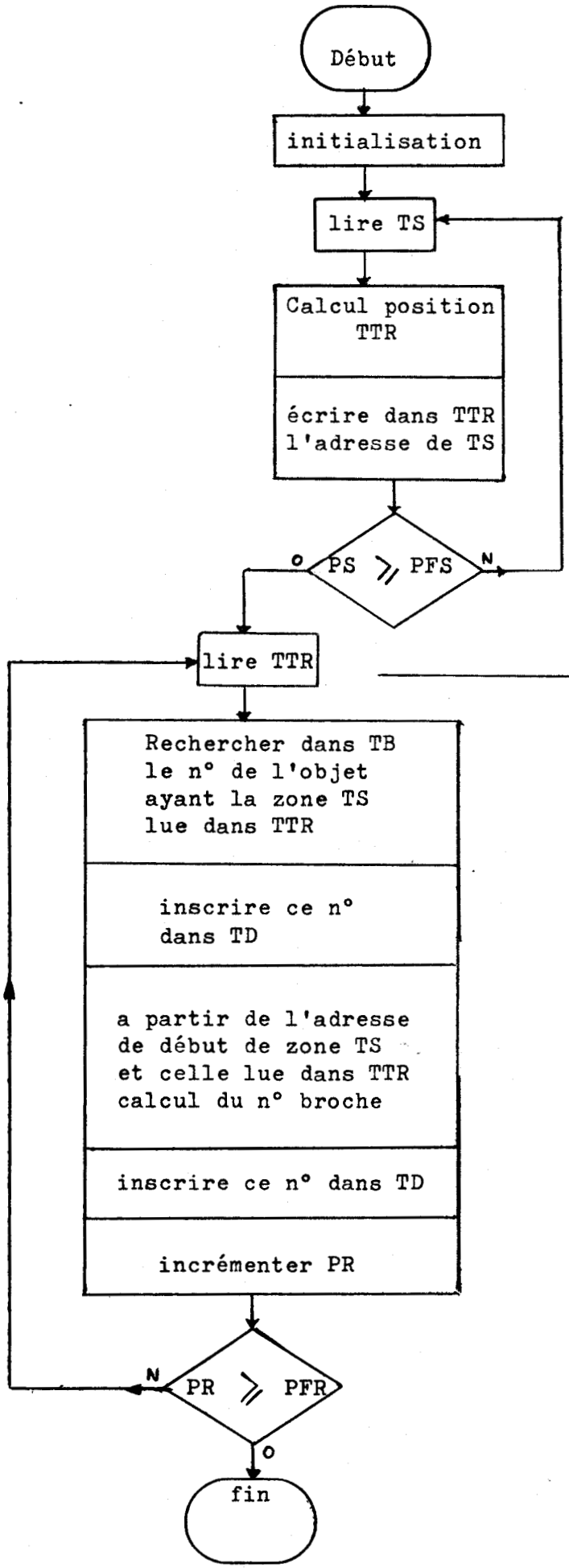
Nous aurons ici aussi trois pointeurs:

- PDD, pointeur de début de table
- PD , pointeur courant
- PFD, pointeur de fin de table.

Pour créer la table destination il suffit de se rendre compte qu'une adresse de la table source permet de retrouver le nom et le n° de broche du signal destination de son contenu. Il suffit de retrouver dans la table de base le signal qui a l'adresse en question comme adresse de zone TS. Le n° de broche étant donné par l'écart entre l'adresse de zone et l'adresse effective.

L'ordinogramme correspondant à ce qui vient d'être dit et décrivant donc la création de la table destination est donné à l'annexe 2.2.

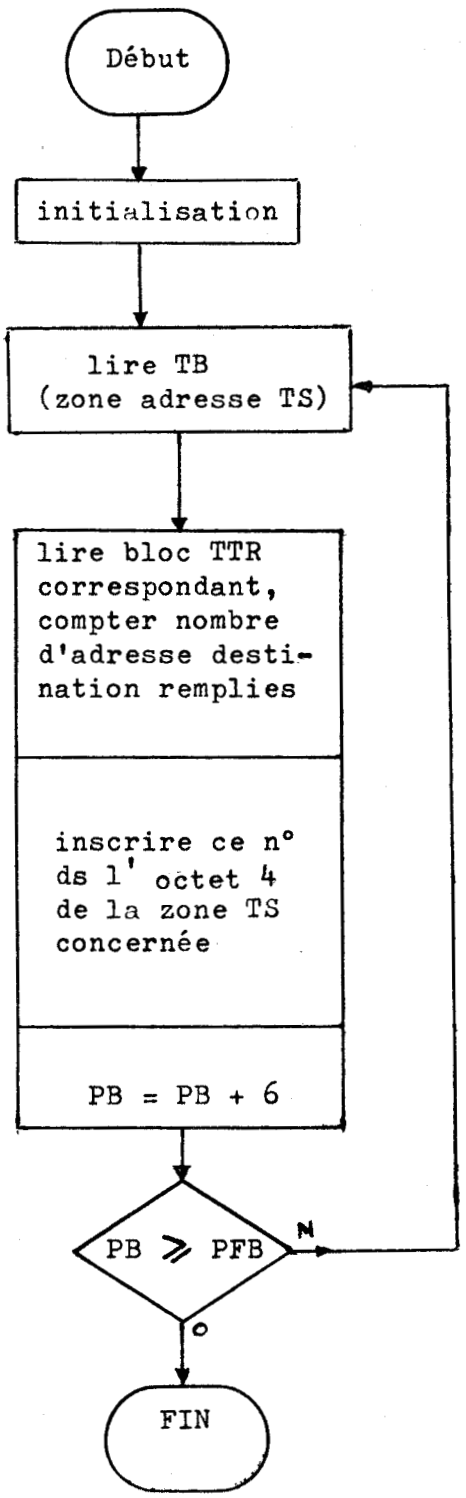
Notons que nous sommes amenés, afin de ne lire qu'une seule fois TS, à créer une table temporaire que nous nommerons TTR et qui contiendra autant de zones qu'il y a d'objets dans la table TB.
A chaque objet de TB correspondra la zone de TTR de même numéro. Chaque zone de TTR comprendra le nombre d'octets nécessaires pour contenir un nombre prédéterminé d'adresses, par exemple 10. Son utilité ressort de l'ordinogramme suivant.
Notons qu'elle sera initialisée en y inscrivant partout la valeur H'FF'.



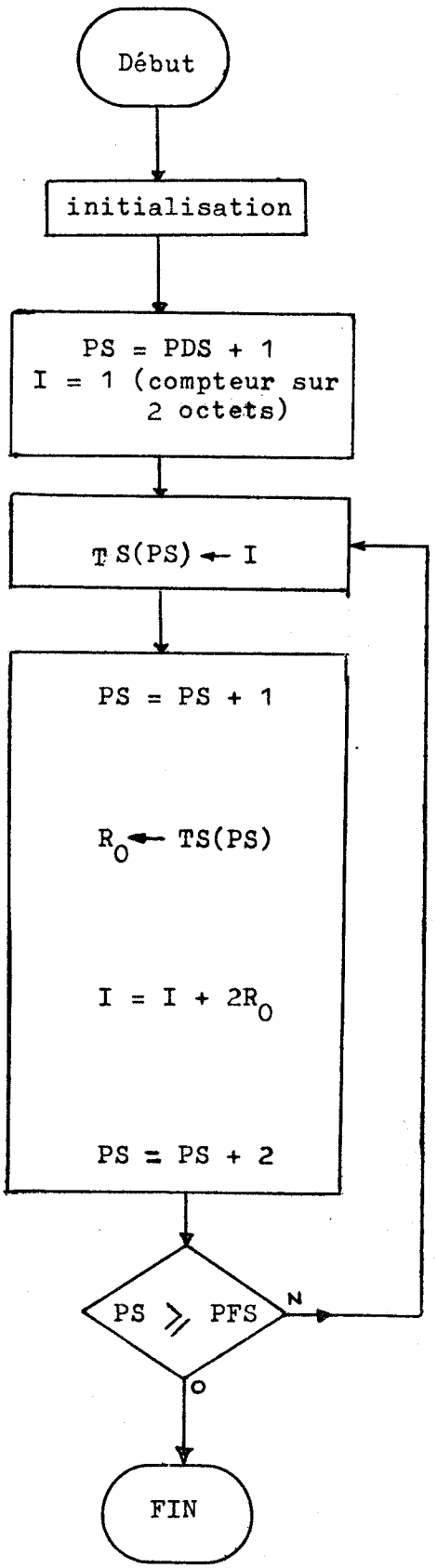
(exemple: si objet n° 4 trouvé en TS → prendre dans TTR la 1e place libre du 4e bloc.)

(on y trouve l'adresse de la zone TS de l'objet de TB destination du signal qui a même rang dans TB que le bloc de TTR lu.)

Nous pouvons maintenant procéder à la 2è passe pour TS



3e passe pour TS



Notons, en terminant ces "tables-guides", qu'au départ nous nous proposions d'en créer trois et qu'en fait nous en avons créer une de plus soit TSPEC. Cette quatrième table-guide est due au langage PHPL qui permet l'appel de modules.

Plus on étendra les possibilités du langage de description, plus la structure des tables-guides deviendra complexe.

II. 5. 2. Evaluation.

a) Entrée des signaux à tester:

Nous disposons actuellement du modèle du circuit à simuler mais avant de pouvoir tester son comportement, il nous faut encore disposer des séquences de signaux aux entrées.

C'est ici principalement que le caractère interactif du système doit intervenir pour ce qui concerne la simulation et il devra présenter les caractéristiques suivantes:

1- Introduction aisée des séquences de signaux

2- Possibilité de visualiser n'importe laquelle des séquences déjà introduites en machine.

3- Possibilité de procéder à l'évaluation du circuit à partir de n'importe laquelle des séquences introduites. Donc possibilité de réutiliser plusieurs fois la même séquence sans devoir la réintroduire.

4- Possibilité de modifier une séquence sans devoir la réintroduire entièrement.

5- Les séquences introduites en machine doivent être mémorisées directement de manière exploitable par l'interpreteur.

Pour pouvoir répondre aux caractéristiques précédentes il faudra avant tout identifier la séquence introduite, aussi envisageons nous la procédure suivante:

- Après appel du programme "introduction de données" la machine demande : - nom du circuit ?
  - nom de la séquence ?
  - est-ce la 1er séquence de ce circuit ?

  Si c'est la première séquence pour le circuit la machine demandera: - nombre de lignes d'entrées?

- La machine visualise alors sur l'écran une colonne de numéros allant de 1 au nombre de lignes précédemment rentré, place le curseur en face du n° 1 et demande à l'utilisateur de rentrer les noms des signaux en concordance avec le n° des lignes.

- La machine demande ensuite à l'utilisateur d'introduire le nombre de phases de la séquence et la durée de chacune d'elles en indiquant la valeur de l'intervalle unitaire et en veillant a ce qu'il soit compatible avec les délais des fonctions et modules utilisés dans le circuit.

- Enfin la machine indique à l'utilisateur qu'il peut introduire les états de chacunes des lignes pour chacunes des phases successivement.
  Cette introduction se fait à l'aide d'un clavier spécial constitué d'une ligne d'interrupteurs (8 interrupteurs correspondants à une porte d'entrée du microprocesseur).
  Chaque interrupteur porte un numéro d'ordre concordant avec le numéro correspondant de ligne défini précédemment.
  Chaque interrupteur pourra alors être positionné par l'utilisateur de manière à introduire un "1" ou un "0". Une touche spéciale permettra de valider les "valeurs" actuelles des interrupteurs.

Dès que le nombre de "phases" introduites sera égal au nombre de phases prévues, la séquence introduite sera automatiquement visualisée.
L'ordinogramme de cette phase de la procédure est reprise à l'annexe n° 2.3.
La visualisation se fera en présentant sur l'écran une ligne de tirets verticaux (I) espacés régulièrement et dont l'écart correspond à un intervalle de temps unitaire.
Le n° des phases est mentionné au dessus du tiret les commençant.
Sur la gauche de l'écran et en dessous de la ligne précédente apparaîtront les noms des signaux et à leur droite, sur deux lignes, sera reprise leur valeur pour l'instant correspondant: la ligne inférieure pour les "0" et la ligne supérieure pour les "1".
A tous moments de cette phase de travail il sera possible de demander la visualisation d'une séquence quelconque, préalablement introduite, en entrant au clavier la commande SIV (Simulation, Introduction de séquences d'entrées, Visualisation) : La machine demande alors le nom de la séquence et la visualise une fois la réponse obtenue.
Pour modifier une séquence, il suffira de rentrer la commande SIM : La machine demandera le nom de la séquence concernée, puis le nom des lignes et le numéro des phases ou il y a un changement a opérer. La procédure est alors semblable à celle correspondant à l'introduction d'une nouvelle séquence.
Naturellement le nombre de séquences possible sera limité et dépendra de la place mémoire disponible.
La technique d'implantation de cette partie étant analogue à celle utilisée pourl'"Editeur" nous nous abstiendrons de produire les ordinogrammes correspondant.
Il nous reste à présent à voir comment sont mémorisées ces informations.

Nous sommes en fait amené à créer quatre tables:

TIS - La table comportant le nom du circuit, le nombre de lignes d'entrée et leurs noms.
Le nombre de séquences introduites pour ce circuit.

TNS - La table comportant les noms des signaux

TPS - La table comportant pour chaque séquence le nombre de phases et la durée de chacune d'elles.

TVS - La table comportant pour chaque séquence la valeur des signaux pour chacune des phases.

Structure de la table TIS:

1 - PDI : pointeur de début de table (fixe)
2 - PFI : pointeur de fin de table (variable)
3 - Les quatre premiers octets seront réservés aux noms du circuit en code ASCII
4 - Le 5e octet contiendra le nombre de lignes d'entrées
5 - Le 6e octet contiendra le nombre de séquences déjà introduites
6 - Les octets suivants seront répartis en groupes de trois octets correspondant aux noms des entrées (en code ASCII) dans l'ordre.

initialisée par la valeur H'OO'

Structure de la table TNS:

1 - PDN : pointeur de début de table = PFI + 1
2 - PFN : pointeur de fin de table = PDN + 60
3 - La table est répartie en groupes de six octets:
    - les deux premiers octets contiennent le nom de la séquence en code ASCII
    - les deux suivants contiennent un pointeur vers la table TPS.
    - les deux derniers contiennent un pointeur vers la table TVS

La table est initialisée avec la valeur H'OO'.

Structure de la table TPS:

1- PDP : pointeur de début de table = PFN + 1

2- PFP : pointeur de fin de table = PDP + 100
(si l'on prévoit une moyenne de dix phases par séquence).

3- Chaque zone de la table correspond à une séquence.
Les pointeurs de la table **TNS** définissent le 1e octet de la zone pour la séquence correspondante.
Ce 1e octet donne le nombre de phases que comprend la séquence.
Les octets suivants donnent la "longueur" de la phase: c'est à dire le nombre d'intervalles de temps unitaires que dure la phase.

Structure de la table TVS:

1- PDV : pointeur de début de table PDV = PFP + 1

2- PFV : pointeur de fin de table PFV = PDV + 100
(la longueur de cette table étant la même que celle de la précédente).

3- Cette table est également divisée en zones correspondant chacune à une séquence.
Les pointeurs de la table TNS définissent également le 1e octet de la zone pour la séquence correspondante.
Ce 1e octet donne également le nombre de phases que comprend la séquence. Les octets suivants eux donnent les valeurs de chacun des signaux d'entrée pour chacune des phases: ce sont les octets rentrés par l'intermédiaire des interrupteurs dont on a parlé précédemment.

Notes: 1) Les tables TVS et TPS pourraient être confondues, mais la capacité mémoire gagnée ne semble pas en valoir la peine: la programmation serait plus compliquée.

2) Remarquons que les 4 tables précédentes utilisent une zone mémoire continue. Cette zone aura été préalablement initialisée avec la valeur H'00'.

b) L'interprétation:

En principe nous avons maintenant toutes les informations nécessaires en place et nous sommes prêts à tester le circuit pour l'une des séquences d'entrée introduite précédemment.

Nous rappelons que nous procéderons à l'évaluation en utilisant une variante de la technique dite de synchronisation sur "l'événement suivant", variante permettant l'introduction de délais différents pour chaque fonction.

Nous allons donc devoir créer les piles LIFO nécessaires, mais cela revient essentiellement à prévoir d'une part, une zone mémoire disponible dont on initialise éventuellement le contenu, et d'autre part, à répartir judicieusement les pointeurs de ces piles en fonction de leur nombre.

Toutefois il y a encore lieu de créer trois tables:

1- La table des délais (TDF)

Cette table donnera, en un octet, le délai introduit par la fonction ou le module correspondant.

Pour chaque fonction ou module le délai correspondant sera obtenu en lisant la table à l'adresse A_X, avec A_X = PD + n° code de la fonction ou module

PD = pointeur de début de table.

2- La table des valeurs des signaux internes au circuit (TVC)

Nous avons une table nous donnant les valeurs des signaux d'entrées mais nous avons évidemment besoin d'une table mémorisant les valeurs de chaque signal du circuit.

Cette table sera constituée d'un nombre d'octets égal au nombre d'objets de la table TB divisé par 8.

En effet, il suffira d'un bit par signal.

Pour trouver le bit correspondant au signal n° X il faudra procéder comme suit:

$$AX = PD + \frac{X}{8} \quad \text{valeur entière}$$

le reste de la division $\frac{X}{8}$ donne la position du bit.

3- La table des valeurs des signaux de sorties (TVS)

Cette table contiendra autant de blocs qu'il y aura de Δt unitaire nécessaire pour donner la réponse à la séquence.
Chaque bloc comprendra autant d'octets qu'il y aura de sorties au circuit divisé par 8.

Nous pouvons à présent envisager la procédure d'évaluation comme suit:

1) Lire TB et y rechercher les entrées du circuit, comparer avec TIS et établir dans une table temporaire TT, la correspondance entre chaque bit d'un octet de TVS et le n° objet de la table TB:

   L'adresse d'un octet de TT relativement à l'adresse de départ de TT, est donnée par le rang du bit de TVS et le contenu correspondra au n° d'objet de TB.

2) Poursuivre la lecture de TB et l'établissement de TT en comparant cette fois avec TDF, la table des délais, afin d'établir une correspondance directe entre le n° objet de la table TB et le délai de la fonction qui y est attaché.

3) Calcul de Δt et du nombre de piles.

4) Créer les n pointeurs de piles, initialiser le compteur de Δt.

5) Positionner l'indicateur de pile d'activités en cours (Sa) sur S1.

6) Empiler dans "Sa" toutes les entrées du circuit.

7) Désempiler "Sa", vérifier la fonction de l'élément dans TB, si c'est une entrée du circuit prendre sa nouvelle valeur dans TVS, la comparer à sa valeur dans TVC, s'il y a changement de valeur on écrit la nouvelle valeur dans TVC et l'on empile les signaux destination dans les "Si" convenables à l'aide de TD et de TDF. Si la valeur n'a pas changée on continue. Si ce n'est pas une entrée du circuit, on cherche sa nouvelle valeur en appelant et en exécutant la sous-routine correspondant à sa fonction, puis on poursuit la procédure comme dans le cas d'une entrée de circuit. Si c'est une sortie de circuit il faut en plus écrire la nouvelle valeur dans TVS 2.

8) Si "Sa" est vide, on prend la première des piles suivantes non vide qui devient "Sa" on incrémente en conséquence le compteur de Δt et on retourne en 7. Si "Sa" n'est pas vide on retourne en 7 directement. Si toutes les piles sont vides on va en 9.

9) Si toutes les phases de la séquence ont été évaluées alors on a terminé. Sinon on passe dans la table TVS à la phase suivante de la séquence et en tenant compte pour choisir "Sa" de la longueur de la phase qui vient de finir. Réajuster le compteur de Δt s'il le faut.
On recommence en 6.

c) Sortie des résultats:

Les résultats seront visualisés en procédant exactement de la même façon que pour la visualisation des séquences d'entrée. On fera suivre les signaux d'entrée par les signaux de sortie du circuit en se servant de TVS 2. La machine indiquera également le nombre atteint par le compteur de Δ t.

II . 6. CONCLUSION.

Nous avons au stade actuel du travail, une description fonctionnelle dont le bon fonctionnement a été éprouvé. Nous pouvons donc l'accepter définitivement.

Mais il faut à présent la concrétiser, ce que nous allons faire au chapitre III qui en fera une transposition en schéma matériel.

Notons que nous ne repartirons plus de la description fonctionnelle (que nous utiliserons encore) mais bien des tables-guides et essentiellement de la table de base.

C H A P I T R E III

T R A N S P O S I T I O N E N

S C H E M A M A T E R I E L

III. 1. INTRODUCTION.

Le but de ce chapitre est de passer de la description fonctionnelle au schéma matériel.

Le premier problème posé sera le choix des composants et donc d'une technologie de réalisation. Comme nous l'avons déjà indiqué, nous ne nous interesserons qu'aux technologies électroniques.

III. 2. CHOIX DE LA TECHNOLOGIE ET DES COMPOSANTS.

III. 2. 1. Structure des données permettant le choix.

Ici aussi l'interactivité permettra de faciliter grandement la tâche. La quantité d'informations relatives aux divers composants peut devenir rapidement très importante . De plus, les choix qui doivent être fait sont parfois quelque peu ambigus. Aussi il ne semble pas possible de pouvoir laisser la machine seule exécuter ce travail. Les possibilités d'un petit système seraient vite dépassées et les résulats peu certains.

Nous adopterons ici aussi une stratégie de questions-réponses. Questions posées par la machine et menant pas à pas vers un choix précis des composants à utiliser.

Mais le choix de l'opérateur ne devra pas être immédiat. il lui sera en effet possible de faire chaque fois appel à une série d'informations avant de se décider.

Les informations à mémoriser dépendront certainement de conditions locales, mais la structure de la base de données qu'elles constituent pourra être étudiée à partir des renseignements essentiels qu'il y a de toute façon lieu de mémoriser. Ainsi, en électronique par exemple, il y a lieu de savoir qu'il existe des composants en technologie MOS ou TTL, qu'en TTL il y a la TTL rapide, la TTL à faible consommation, etc. Pour les fonctions qui existent dans l'une ou l'autre catégorie , nous devrons connaître entre autres, leurs caractéristiques électriques, mécaniques, leurs prix et l'état du stock.

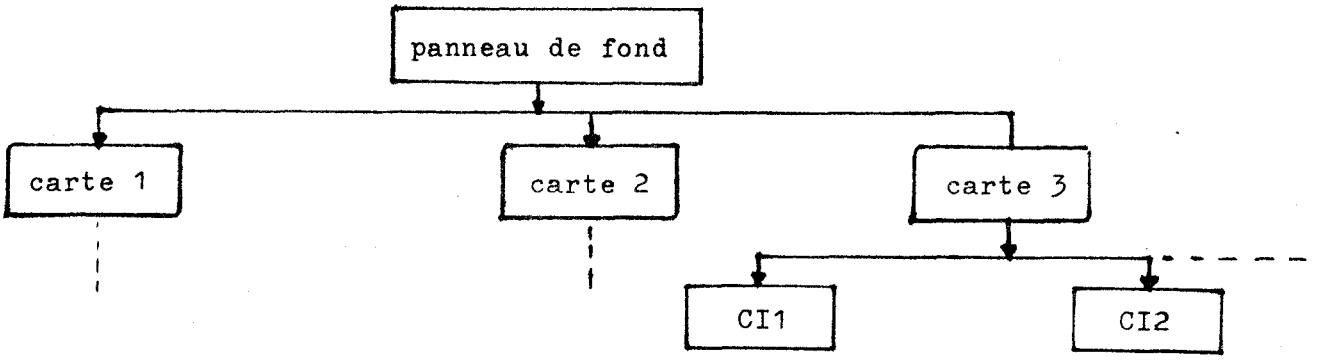
Comment organiser les fichiers de toutes ces informations ?

Si nous nous en référons à Dan NASH (1) il semble que nous ayons essentiellement trois modèles de bases de données:

a) Le modèle hiérarchique:

Dans ce modèle les fichiers de données sont organisés suivant une structure en arbre hiéarchisée.

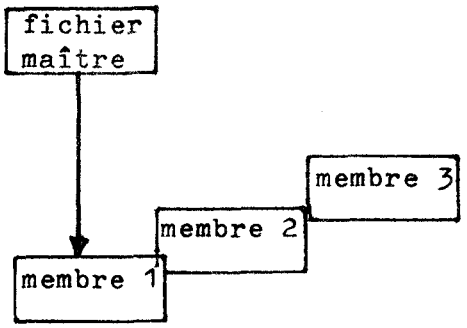
Ce type d'organisation apparaît dans beaucoup de systèmes. Prenons par exemple un système digital composé d'un panneau de fond réalisé en cablage matricé (wire wrap) et interconnectant un certain nombre de cartes imprimées, ces cartes étant elles mêmes composées de circuits intégrés, etc. Avec un nom de système comme clé nous pouvons accéder dans la base de données correspondante et traverser l'arbre de façon à obtenir les données voulues:



b) Le modèle Codasyl

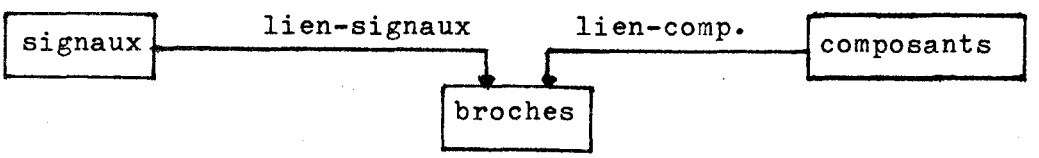
Ce modèle est basé sur le rapport CODASYL de 1971 (ACM).

Il est essentiellement basé sur la notion de groupe de fichiers. Un groupe de fichiers est constitué d'un fichier "maître" qui peut avoir un nombre variable de fichiers "membres" auxquels il est relié par un "nom de groupe". La puissance de ce modèle provient du fait qu'un "maître" peut avoir n'importe quel nombre de "membres" , un "membre" n'importe quel nombre de "maîtres" et un membre dans un groupe peut-être "maître" dans un autre.



* maître directement accessible par une clé
* membres accessibles uniquement par l'intermédiaire du maître.

Exemple: Soit un fichier "signaux" , un fichier "composant" et un fichier "broches".
Avec une structure de type CODASYL:

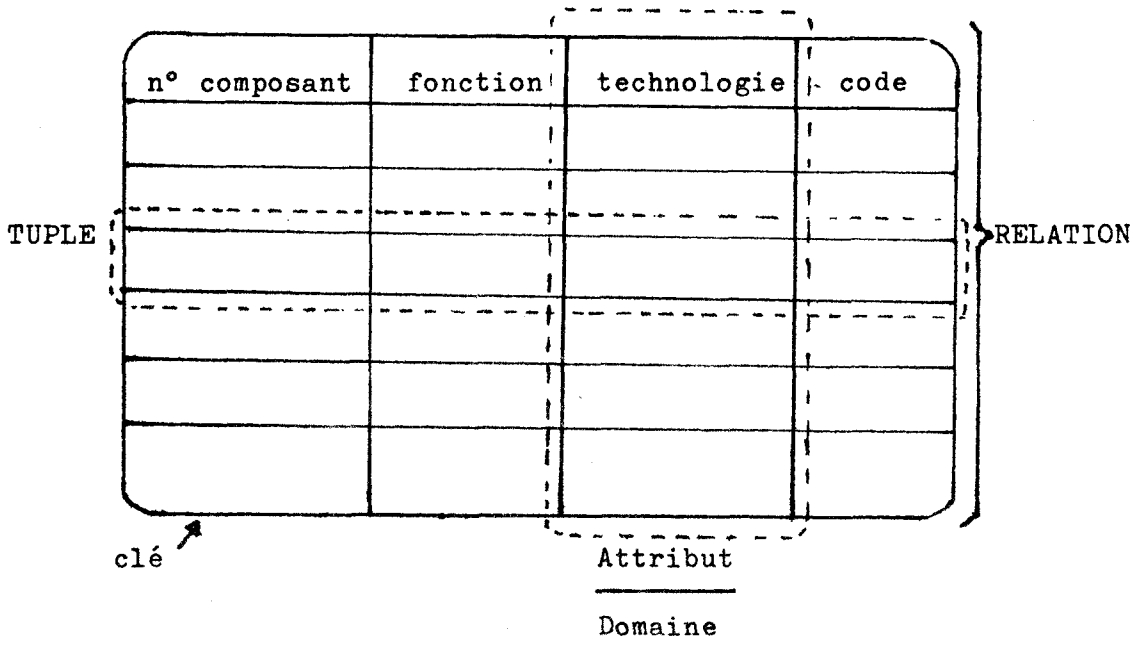


En donnant le nom d'un signal nous pouvons avoir toutes les broches de l'équipotentielle. En donnant le nom du composant nous pouvons avoir toutes les broches le concernant.

c) Le modèle relationnel

Les deux modèles précédents sont basés sur une technique de "navigation" d'un fichier à l'autre. Ce modèle ci, proposé par E.F. Codd en 1970, tente d'établir une structure de relation entre les données sans qu'il soit nécessaire de créer des "chemins" les reliant.

En fait il s'agit de simples tables a deux dimensions:



Chaque table est une relation. Chaque ligne, qui définit un vecteur de données est un tuple. Chaque colonne, qui spécifie un type de valeur est un attribut. L'ensemble des valeurs permises dans une colonne est le domaine. Chaque tuple est adressé par une clé unique.

Les propriétés importantes d'une relation sont:

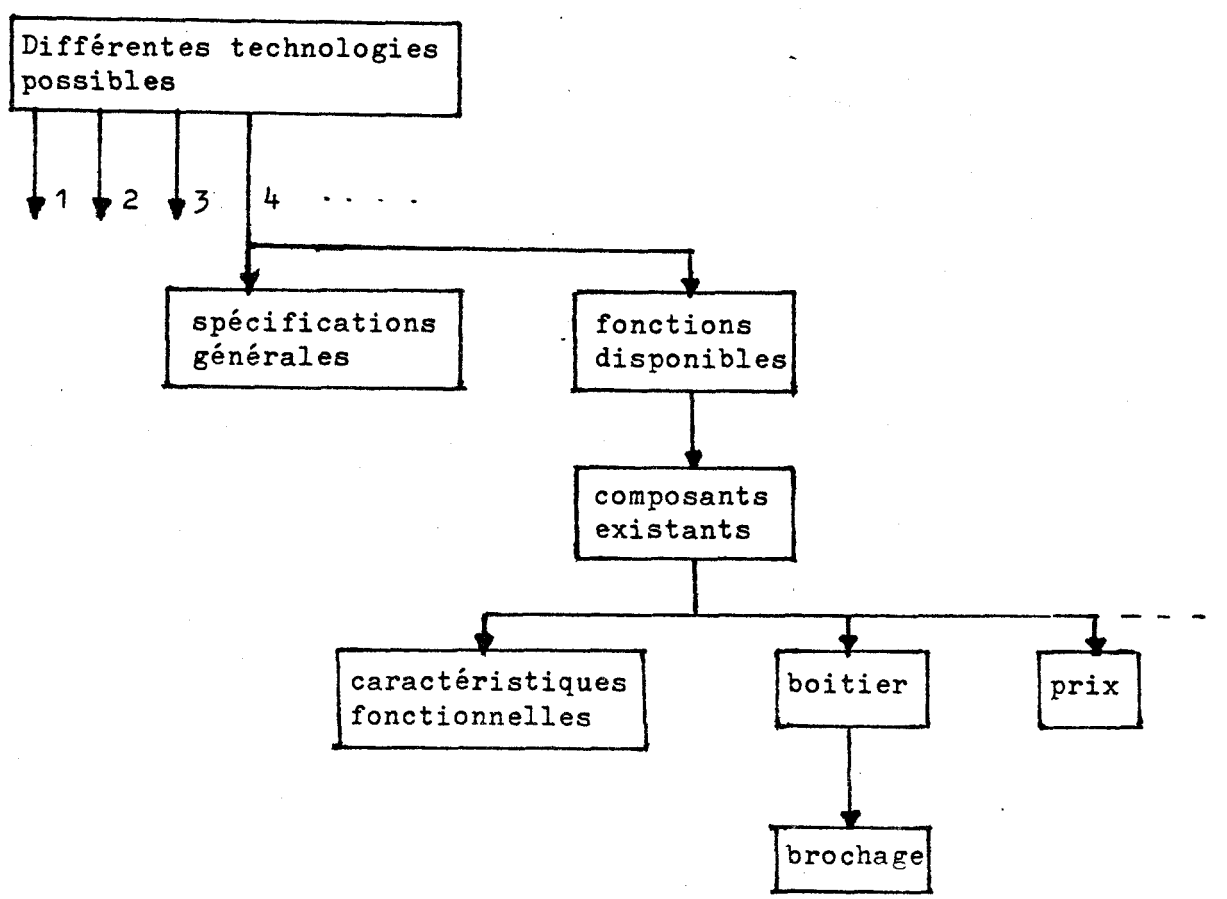
- il n'y a pas deux lignes identiques
- l'ordre des lignes n'est pas significatif
- l'ordre des colonnes est significatif.

Plusieurs relations forment ensemble une base de données relationnelles.

Choix du type de base de données:

Il n'est pas impossible qu'une base de données sur l'un ou l'autre des deux derniers modèles ne s'avère plus rentable dans le cas d'un gros système. Toutefois un modèle hiérarchique s'adapte parfaitement à la technique questions-réponses du mode interactif et est simple à implanter.

Nous envisageons donc une organisation selon le schéma suivant, très semblable en fait à celui prévu pour les fonctions et modules au premier chapitre:



III. 2. 2. Procédure menant au choix de la technologie.

Cette procédure sera fort semblable à celle de l'éditeur:

Après appel du programme de "transposition matérielle" la machine demande:

- Quelle est la technologie choisie ?

Pour cela elle présente la liste des technologies possibles et place un curseur clignotant en regard de la première citée. Elle abaisse le curseur tant que l'opérateur n'a pas répondu par oui (O). Si l'opérateur répond oui, la machine pose la question:

- Est-ce un choix définitif ?

Si la réponse est affirmative, elle pose encore la question:

- Pour toutes les fonctions et tous les modules ?

Si la réponse est affirmative elle passe à la phase suivante, c'est à dire au choix précis de chaque composant.

Si la réponse est négative par contre, la machine demande la liste des fonctions et modules pour lesquels l'opérateur choisit cette technologie. Après quoi elle reprend le processus au niveau de la liste des technologies possibles, en placant le curseur au niveau de la technologie suivant celle venant d'être sélectionnée.

Par contre si, à la question précédente (choix définitif ?), la réponse est négative, la machine propose à l'opérateur une liste de fiches fournissant diverses précisions sur la technologie sélectionnée. L'opérateur peut alors suivant la technique habituelle demander la visualisation de l'une ou (et) l'autre de ces fiches. Dès qu'il s'estime suffisament informé l'opérateur le signifie à la machine en entrant un " R " qui ramène la procédure à la liste des technologies possibles, mais cette fois avec le curseur en face du nom de la technologie sélection-

née. L'opérateur peut ainsi confirmer son choix ou non.

La ou les technologies définies, il faut a présent passer au choix des composants.

III. 2. 3. Choix des composants.

Nous allons ici créer la table des composants (TCO) et préparer la table des connexions (TC). Pour cela nous aurons besoin de deux tables temporaires: la table des fonctions et modules nécessaires (TFN) et la table d'affectation temporaire (TAT).

La table TFN constitue en fait le bilan des fonctions et modules nécessaires réalisé à partir de la table de base TB.

Elle sera créée en lisant dans la table de base les 4e et 7e octets de chaque objet. C'est à dire les codes fonctions et le nombre d'entrées de ces fonctions. Un classement de ces codes sera effectué et l'on mémorisera également le nombre de fois qu'une même paire de code sera détectée.

Ce qui donne pour la table TFN la structure suivante:

- un pointeur de début de table (PDN)
- un pointeur de fin de table (PFN)
- un pointeur courant (PN)
- chaque objet de la table sera constituée de quatre octets:
  1e octet : code fonction ou module
  2e octet : nombre d'entrées
  3e octet : nombre de fonctions de ce type dans le circuit.
  4e octet : pointeur vers le complément de la table TFN.

Le complément de la table TFN constitue simplement la liste des n° des objets de la table TB correspondant au type de fonction ou de module concerné. Ce complément de table sera réalisé au cours d'une deuxième passe.

La table TFN sera visualisée sur une colonne à gauche de l'écran. Bien entendu le code fonction sera remplacé par un nom en clair.

Sur la droite de l'écran sera également visualisé la liste des composants possibles en fonction des technologies choisies.

L'opérateur pourra alors demander de visualiser, au centre de l'écran, les caractéristiques désirées des composants qui l'intéressent, ainsi que la description fonctionnelle du circuit afin de procéder à certaines affectations particulières.

En effet, il se peut, par exemple, que l'opérateur remarque parmis les composants possibles, un bistable JK dont les entrées J et K sont précédées, dans le même composant, d'une fonction ET à trois entrées. En vérifiant la description fonctionnelle l'opérateur peut aussi constater que l'utilisation d'un tel composant, plutôt que des composants séparés pour les bistables et les fonctions ET, en certains endroits du circuit au moins, permet de diminuer les connexions externes et le nombre de composants utiles. De fait, il arrive parfois qu'une seule fonction ET demande l'utilisation d'un composant en contenant trois ou quatre.

Ce type de recherche sera faite plus aisément par l'opérateur que par la machine, surtout si celle-ci lui fournit toutes les informations nécessaires.

Le choix de ces composants spéciaux étant fait, ainsi que leur affectation, l'opérateur pourra les rentrer en machine de manière analogue à l'édition de la description fonctionnelle.

Il rentre au clavier:

- le nom du composant
- la liste des signaux sur chaque broches (dans l'ordre croissant des broches)
- un signe de séparation (par exemple : $)
- la liste des fonctions ou modules qui viennent d' être affectés
- un signe "next" si l'on désire introduire un autre composant de manière similaire.
- un signe "Ret" si l'on a terminé d'introduire les composants spéciaux.

La machine lit alors la zone tampon contenant le texte rentré et procède à l'établissement de la première partie de la table TAT: elle y place le nom des composants retenus, chacun de ces noms étant suivis de la liste des n° des objets de la table de base qui y sont affectés. La table TFN est actualisée.

La structure de la table TAT est la suivante:

- pointeur de début de table (PDA)
- pointeur de fin de table (PFA)
- pointeur courant (PA)
- H' FF'
- nom ASCII du composant
- H' 11'
- liste des objets de TB qui y sont affectés
- H' FF'

La machine construit simultanément le début de la table TCO, suivant la même structure, mais en n'y plaçant que les noms des composants. La suite de cette table TCO est construite par la machine en comparant la table TFN réajustée à la liste des composants possibles. Notons que l'on différencie les composants identiques par un numéro d'ordre.

Il nous reste à présent à "remplir" ces composants et à affecter un signal à chaque broche , ce qui revient donc aussi à créer la table des connexions (TC).

III. 3. AFFECTATION DES FONCTIONS AUX DIVERS COMPOSANTS.

En réalité il y a encore deux stades à franchir pour pouvoir déterminer les connexions broche à broche :

1) affectation des fonctions aux différents composants
2) répartition des fonctions dans le composant et affectation des broches d'entrées et éventuellement de sorties.

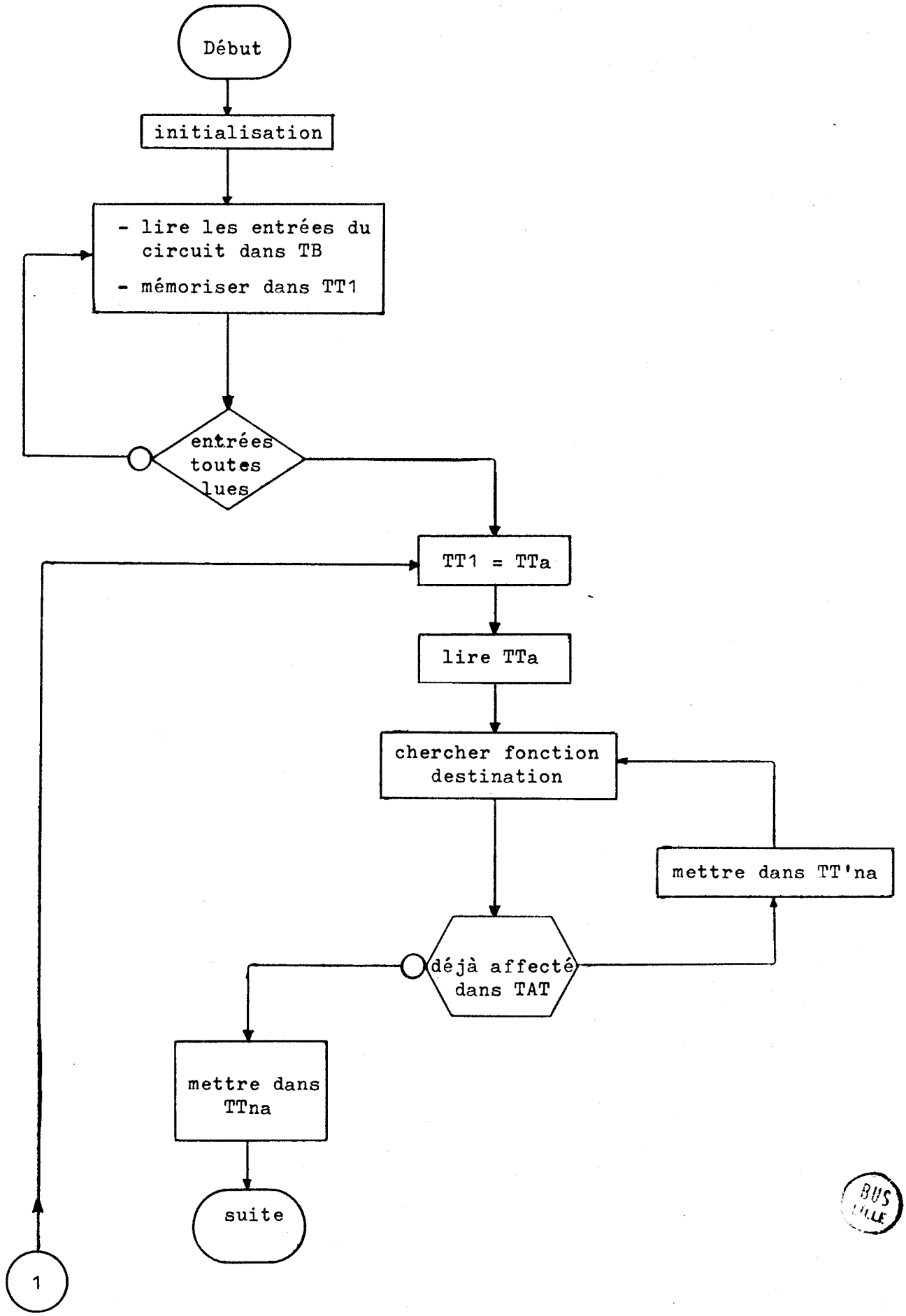
Le premier stade peut être franchi dès à présent, le second lui se fait normalement après le placement des composants sur la carte de circuit imprimé, diverses techniques sont mentionnées par David W. Hightower (2), une technique particulièrement interessante est développée par Leah Mary-Rauch (3), toutes cependant se basent sur la disposition des composants sur la carte.

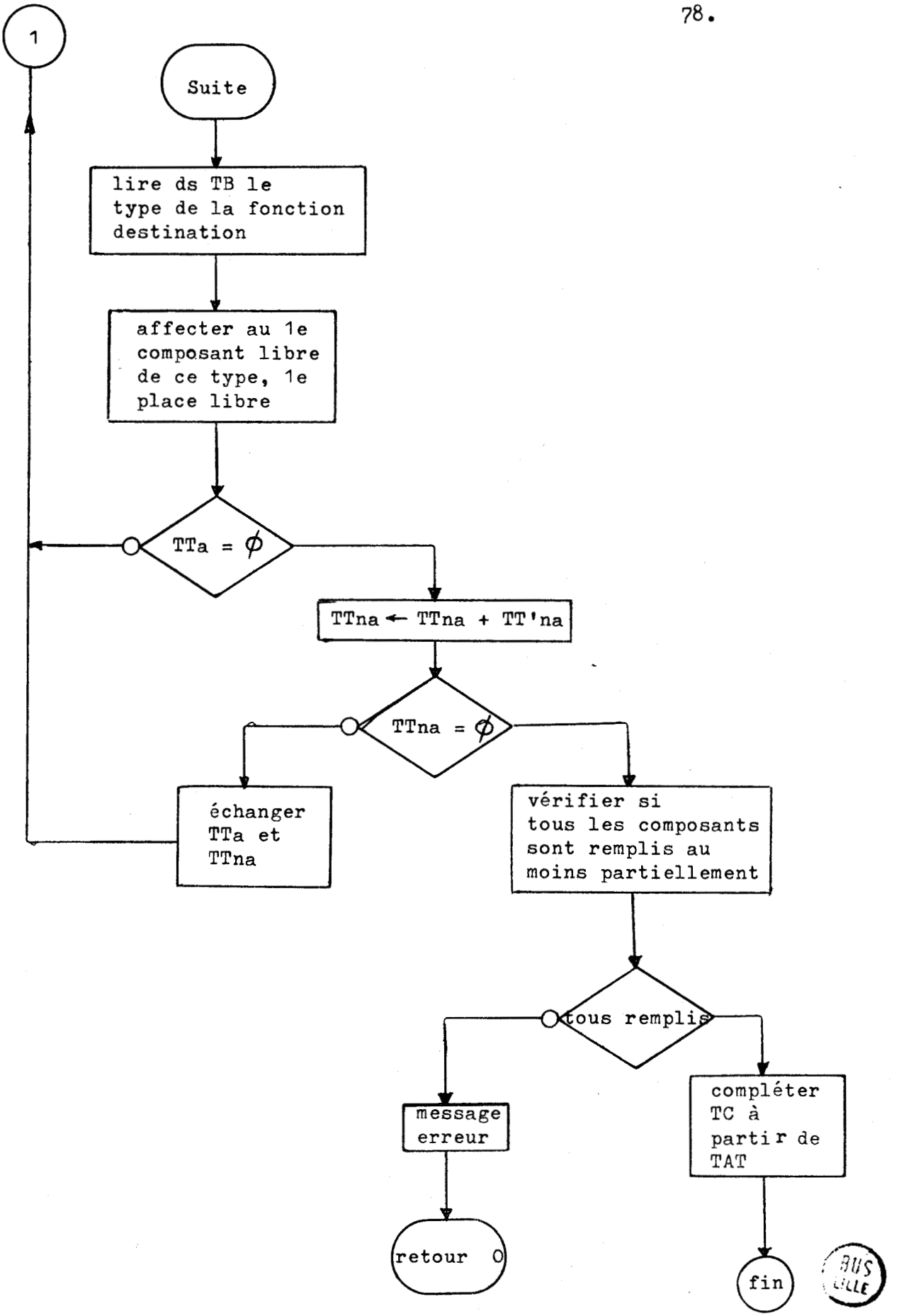
Nous affecterons néanmoins arbitrairement et temporairement chaque fonction à une place déterminée, dès à présent,il en sera de même pour les broches d'entrées.

Nous aurons ainsi une structure de données complète et définitive. Il suffira après le placement des composants sur la carte imprimée a procéder à de simples échanges de données ou de groupes de données.

Revenons maintenant à l'affectation des groupes de fonctions aux différents composants. Nous utiliserons la procédure suivante qui tend à placer dans un même composant les fonctions de même types qui semblent les plus proches les unes des autres,lorsque l'on suit les chemins des différents signaux,en parallèle,à partir des entrées.

La procédure est alors simple et est décrite par l'ordinogramme suivant:


Début
initialisation
- lire les entrées du circuit dans TB
- mémoriser dans TT1
entrées toutes lues
TT1 = TTa
lire TTa
chercher fonction destination
mettre dans TT'na
déjà affecté dans TAT
mettre dans TTna
suite
1


1
Suite
lire ds TB le type de la fonction destination
affecter au 1e composant libre de ce type, 1e place libre
TTa = ϕ
TTna ← TTna + TT'na
TTna = ϕ
échanger TTa et TTna
vérifier si tous les composants sont remplis au moins partiellement
tous remplis
message erreur
retour 0
compléter TC à partir de TAT
fin

L'affectation des fonctions se concrétise en remplissant la table des connexions (TC) dont la structure est décrite ci-après.

Comme nous l'avons déjà signalé, la répartition des fonctions dans les composants et l'affectation des broches se fera arbitrairement. Nous faisons en fait correspondre l'ordre de numérotation des broches du composant avec l'ordre des signaux dans la table TS.

La structure de la table des connexions (TC) sera la suivante:

1) PDC : pointeur de la table TC
2) PFC : pointeur de fin de table TC
3) chaque nouveau composant dans la table est précédé d'un octet contenant la valeur H' 11'.
   Notons que H' 11' sert de frontière entre deux composants décrits.
4) H' 11' est suivi du code ASCII du composant dont la description des connexions broche par broche suit.
   Si l'on désire retrouver le type de composant correspondant il faut évidemment se référer à la table des composants (TCO)
5) H' FF' suivra le code ASCII précédent et se réprésentera de manière a servir de frontière entre les descriptions des connexions de chacune des broches.
6) H' FF' sera suivi d'un octet contenant le n° de la broche dont on va donner les connexions.
7) Ce numéro sera suivi d'autant de paire d'octets qu'il y a de connexions partant de la broche considérée.
   Le premier octet de cette paire donne le code ASCII du composant vers lequel part la connexion décrite.
   Le deuxième octet spécifie le n° de la broche ou arrive la connexion.

III. 4. CONNEXIONS COMPLEMENTAIRES PROPRES AU MATERIEL.

Avant tout dans cette partie la machine établira seule les connexions d'alimentations. Il suffit qu'elle lise dans les caractèristiques des composants choisis les diverses tensions nécessaires et les broches qui leurs sont affectées. Elle pourra alors compléter TC en conséquence.

Ensuite il pourra y avoir les broches non utilisées et qui devront être connectées d'une façon ou d'une autre. Ainsi si nous reprenons l'exemple du bistable JK précédé d'une porte ET à trois entrées, il est possible que seulement deux entrées soient nécessaires, la troisième doit alors être reliée à la tension correspondant au 1 logique.

Ce genre de connexion sera vraisemblablement réalisé plus facilement par l'opérateur si on lui fournit clairement les informations utiles.

Pour cela la machine visualisera les composants en indiquant convenablement mais simplement les broches affectées et celles qui ne le sont pas.

L'opérateur pourra alors visualiser également les différentes caractéristiques des composants, la table des connexions, la description fonctionnelle et introduire alors manuellement les connexions voulues.

Notons avant de terminer cette partie qu'il y aurait lieu de procéder à la vérification du schéma matériel obtenu et à sa simulation. Il faut donc à partir des données sous leur forme actuelle pouvoir reconstituer les tables de base , source et destination. Il faudra aussi probablement créer des nouvelles sous-routines rendant compte des caractéristiques de certains composants.

III. 5. NOTE SUR LA POSSIBILITE D'UNE IMPLANTATION DE LA DESCRIPTION FONCTIONNELLE DANS UN MODULE PROGRAMMABLE.

Au début de ce chapitre, lors de la procédure du choix de la technologie une première question aurait pu poser le choix entre une implantation cablée ou programmée. Nous n'avons parlé en fait que de la solution cablée.

Le développement du système pour les solutions programmées est évidemment possible. La structure générale pourrait être assez semblable, le choix des composants, qui seraient réduit à quelques unités, devrait se compléter par l'élaboration du programme a y implanter. Toutefois certaines informations nous manquent actuellement. En effet, lors de la simulation, il y aurait en tous cas lieu de vérifier ce qui peut être traité séquentiellement et dans quelles conditions. Nous avons en effet au début de ce travail choisi un langage qui permet la description d'actions parallèles, il faudra voir dans quelle mesure ces actions pourront être réalisées les unes après les autres sans nuire au bon fonctionnement du système. En fait la séquentialisation de ces actions est une forme de délai supplémentaire qui est ainsi introduite. La limitation principale des systèmes programmés est d'ailleurs leur lenteur relative par rapport aux systèmes cablés. Une solution à ce problème peut parfois être obtenue en doublant l'unité de traitement de données. Les avantages d'un système programmé étant bien entendu la facilité que l'on a d'y apporter des modifications, la compacité et le fait qu'un même module matériel peut satisfaire un grand nombre d'applications diverses.

Le problème de la transposition matérielle sera donc essentiellement le choix du module de base, le partage éventuel des tâches entre unités de traitement et le séquencement de ces tâches. Le choix des composants quant à lui deviendra essentiellement un choix de sous-routines qu'il faudra "relier"

III. 6. CONCLUSION.

Nous avons à présent concrétisé la description fonctionnelle de départ en un schéma matériel réalisable physiquement. Encore faut-il placer les composants sur un support et effectuer les connexions prévues. La forme la plus répandue actuellement de solution à ce problème dans les réalisations industrielles est la carte de circuit imprimé. L'étude d'une telle carte pour le schéma matériel que nous venons d'obtenir constituera donc l'objet du chapitre IV.

CHAPITRE IV

ETUDE DE LA CARTE DE CIRCUIT IMPRIME.

IV. 1. INTRODUCTION.

Il existe plusieurs systèmes d'aide à la conception de circuits imprimés. Nous n'en citerons que quleques uns pour lesquels nous donnerons les caractéristiques principales.

a) Le système Labyrus (1973) (THOMSON - CSF) (56)

Ce système réalise automatiquement, l'implantation et le tracé de circuits imprimés selon une technologie double couche. Les données du traitement consistant essentiellement en une description de schémas électriques et en directives fournies par l'utilisateur, permettant de respecter les contraintes des réalisations manuelles. Les résultats comprennent les films et les rubans perforés pour la fabrication.

Les programmes principaux sont les programmes PAPYRUS (15) pour l'implantation automatique ou assistée des composants et le programme TRACUS (14) pour le tracé automatique.

L'exploitation de la chaîne nécessite un ordinateur disposant de 230 K octets de mémoire centrale.

b) Procédé RTC (53) (1975)

Ce système est entièrement conversationnel.
Il comporte: - un calculateur PDP de digital Equipement connecté à une console graphique et un téléimprimeur.

- une table traçante automatique, pouvant également être utilisée pour conception par digitalisation.

c) Système Norvégien. (1975-1978) (57)

Ce système utilise les sous-systèmes suivants:

- CASS (Computer Aided Schématic System)
- TEGAS (Test Generation and Simulation System)
- CAMILA (Computed aided Automatic and Manual Layout).

L'aide à la conception apportée par ce système débute donc par l'introduction en mode interactif des schémas à l'aide du système CASS. Toutes les informations relatives au placement des composants, aux interconnexions, aux noms des composants, à l'assignation des broches, aux noms des signaux, etc. sont mémorisées. Dès que l'édition est terminée la matrice topologique est créée et les données relatives aux composants sont recopiées à partir de la bibliothèque du système.

Ensuite vient la phase de vérification en testant la topologie du circuit. Si les informations manquent elles sont introduites interactivement à partir de la console d'entrée. La vérification est faîte par le système TEGAS.

Enfin la procédure de conception de la carte est réalisée par CAMILA qui assure l'assignation des broches, le placement des composants et le tracé des interconnexions. Les deux modes, interactif et automatique, sont possible. Le placement des composants nécessite cependant un placement initial par l'opérateur.

Le système est implanté sur un mini-ordinateur utilisant de 32 à 64 K mots de 16 bits comme mémoire centrale et 2,4 millions de mots sur disques.

Le système dispose également d'une console de visualisation graphique et d'une imprimante.

Pour notre part nous décomposerons ce chapitre en deux parties: la première consiste à placer les composants sur la carte et la seconde à rechercher les chemins réalisant les interconnexions.

La première partie est la plus difficile et demandera donc une participation plus grande de l'opérateur.

La seconde partie pourra au contraire être confiée presque exclusivement à la machine. L'opérateur aura toutefois la possibilité de modifier certains chemins ou d'en supprimer d'autres. Les chemins supprimés pourront être recherchés ultérieurement par la machine ou par l'opérateur: ce qui revient à modifier l'ordre suivant lequel on construit les chemins et peut lever certains refus.

Note: par refus on entend un chemin non trouvé par la machine.

IV. 2. LE PLACEMENT DES COMPOSANTS

IV. 2. 1. Aperçu général du problème:

Le problème du placement des composants revient à rechercher pour chacun d'eux, la position optimale sur la carte dans le but de faciliter au maximum les interconnexions prévues. Ce placement se

fera en respectant une grille de positionnement dont le choix constitue l'une des premières tâches de cette partie du travail.

Cette tâche est normalement réalisée simplement en divisant la carte en autant de surfaces élémentaires qu'il y a de composants à placer. Notons qu'il conviendra de définir une "unité composant". En effet des éléments simples, comme des résistances, pourront être regroupés et considérés comme un seul composant, à l'inverse un circuit intégré 40 broches pourra être considéré comme équivalent à plus d'un composant.

La complexité du problème posé est telle qu'aucune méthode ne permet de garantir un placement optimal des composants sur la carte. Aussi fait-on appel à divers algorithmes heuristiques qui seront soit de type à "amélioration par itérations succesives", soit du type à "construction progressive".

- Les algorithmes du type à "amélioration par itérations succesives", partent d'un placement initial arbitraire des composants sur la carte.

Suivant des règles précises, on échange la position de certains composants en cherchant à améliorer la fonction de coût du placement à chaque étape. La fonction de coût rend compte de la "faisabilité" des interconnexions. En fait, normalement, on recherche à en minimiser la longueur.

- Les algorithmes du type à "construction progressive" réalisent le placement des composants en ajoutant un des composants non encore placé à ceux déjà en place.

Les composants placés ne sont plus changés.

Il y a lieu bien entendu, d'établir des règles de sélection du composant à placer. De même il faut établir des règles permettant

de déterminer sa position sur la carte.

Les règles de sélection recherchent parmis les composants qui ne sont pas encore en place celui qui est le plus interconnecté à ceux déjà positionnés.

Les algorithmes à amélioration par itérations succesives, nécessitent généralement un temps de traitement plus important mais doivent, normalement, conduire à des résultats meilleurs que les algorithmes à construction progressive. Ces derniers sont comparativement beaucoup plus rapides et conduisent à des résultats satisfaisants dans beaucoup d'applications.

Nous donnerons un exemple d'algorithme de chaque type.

L'algorithme dit du "Min-Cut Placement" (7) est à amélioration par itérations succesives, celui proposé par G. ALIA, G. FROSINI et P. MAESTRINI (4), est à construction progressive.

IV. 2. 2. Exemples d'algorithmes.

a) Min-Cut Placement (M.A. BREUER- 1977)

Les algorithmes classiques de placement des composants sur la carte cherchent à minimiser la longueur totale des interconnexions:

$$N_d = \sum_{\forall s} d_s$$

avec d_s = longueur de l'interconnexion s

$$= \Delta x + \Delta y$$

L'auteur a recherché une nouvelle fonction tendant a optimiser le placement des composants sur la carte.

Cette nouvelle fonction lui a été suggérée par deux observations:

- Le succès du tracé des interconnexions dépend de leur densité sur la carte.
- La densité des connexions est plus importante sur certaines parties de la carte que sur d'autres.

1) Définition de la fonction "Min-Cut"

Soit c une ligne coupant la surface de la carte horizontalement (ou verticalement).

Si un ou plusieurs éléments d'une équipotentielle se trouvent au dessus de c et un ou plusieurs autres éléments se trouvent en dessous de c , alors l'équipotentielle coupera au moins une fois la ligne c.

Si L est la largeur en cm de la carte et si w est la densité par cm des connexions pouvant être tracées, alors,pour qu'un placement de composants sur la carte soit acceptable, il faudra que:

$\varphi(c)$, le nombre d'équipotentielles ayant une partie des éléments d'un coté de c et l'autre de l'autre coté, soit inférieur ou égale à L.w , soit:

$$\varphi(c) \leq L.w$$

L'auteur propose un algorithme tendant à minimiser $N_c(\sigma) = \sum \varphi(c)$ pour un ensemble de lignes c à déterminer. Il montre également que si l'on miniminise $N_c(\sigma)$, on minimise également N_d .

2) Principe général de l'algorithme

Soit une carte sur laquelle on dispose arbitrairement les boitiers à placer. On considère que l'ensemble de la carte constitue

un seul bloc B contenant l'ensemble des N boitiers. Au moyen d'une première ligne C on sépare le bloc B en deux blocs B_1 et B_2 contenant respectivement N_1 et N_2 boitiers ($N_1 + N_2 = N$). On miniminisera alors $N_c(\sigma)$ en échangeant certains boitiers entre B_1 et B_2 et cela en suivant, par exemple, la procédure proposée par Kernighan et Lin (27).

En effet le problème posé à ce moment-ci de la procédure est le suivant:

Etant donné un graphe G ayant N noeuds, partager l'ensemble des noeuds de G en deux ensembles distincts B_1 et B_2, ayant N_1 et N_2 éléments respectivement, ou $N_1 + N_2 = N$, et tel que le nombre d'arêtes entre B_1 et B_2 soit minimum.

Ce qui correspond bien au problème traité par Kernighan et Lin entre autres.

Il suffit à présent de recommencer la procédure en partageant B_1 et B_2 eux aussi en deux blocs chacun et ainsi jusqu'a ce qu'il n'y ai plus qu'un boitier dans chaque bloc.

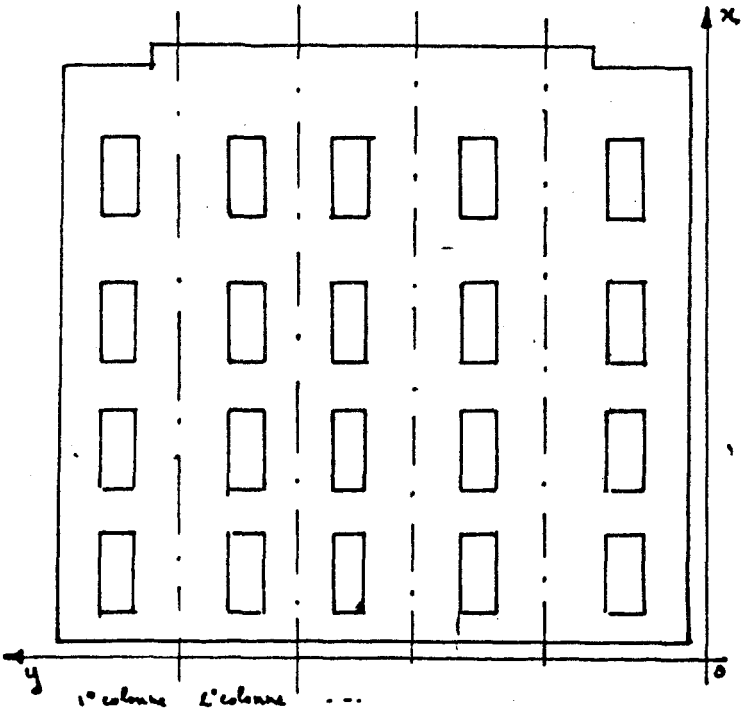
L'auteur présente différentes façons de "découpage" des blocs:

- en quadrature: lignes de coupures succesivement horizontales et verticales.
- en tranches (horizontales ou verticales):
  les N boitiers sont réparts en K et (N-K) boitiers, puis les (N-K) en K et (N-2K) et ainsi de suite.
- en "Bissection" : on répartit chaque fois les boitiers en quantités égales (à 1 près) dans les deux nouveaux blocs créés.

b) Algorithme de placement proposé par G. ALIA, G. FROSINI et P. MAESTRINI (1973) (4)

La carte sera décomposée en une série de colonnes de largeur Δy.

Le connecteur sera parallèle à l'axe y et les boitiers auront leur grand coté parallèle à l'axe X . Chaque colonne comprendra une seule rangée de boitiers, en nombre déterminé.



Le problème sera fractionné en trois sous-problèmes:

1) Répartition des boitiers par colonnes

2) Ordonnancement des boitiers dans chaque colonne

3) Ordonnancement des colonnes sur la carte.

1) Assignement des boitiers dans les colonnes.

Le "remplissage" des colonnes se fait colonne par colonne et, dans chaque colonne, boitier par boitier. Lors du "remplissage" d'une colonne une"fonction de mérite" V_3 est associée à chaque boitier restant à caser. Le boitier ayant le plus grand V_3 est choisi pour venir s'ajouter à la colonne que l'on rempli:

$$V_3 = \frac{1 + V_1}{1 + V_2 + a\eta}$$

a = nombre moyen de liaisons au connecteur de l'ensemble des boitiers déjà dans la colonne considérée.

η = 1 si le fait d'ajouter ce boitier à ceux déjà dans la colonne considérée augmente le nombre moyen de liaisons au connecteur.

$\eta = 0$ dans les autres cas.

V_1 = est le nombre d'interconnexions entre ce boitier et ceux déjà dans la colonne considérée.

V_2 = est le nombre d'interconnexions entre ce boitier et ceux qui restent encore à caser.

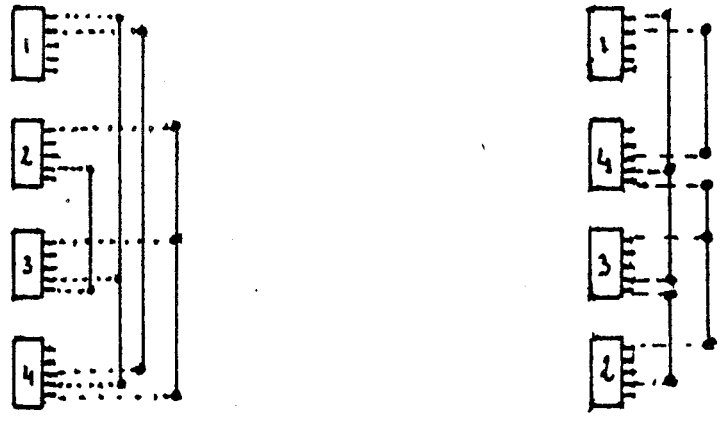
Le 1 au dénominateur évite les divisions par zéro

Le 1 au numérateur permet de distinguer les boitiers qui ont une fonction $V_1 = 0$

Lorsque le nombre prévu de boitiers par colonne est atteint on commence automatiquement le remplissage de la colonne suivante.

2) Ordonnancement des boitiers dans une colonne.

Dans une colonne les boitiers sont rangés de façon à minimiser le nombre de lignes verticales occupées (// à x) pour effectuer les interconnexions entre boitiers, voir figure ci dessous: de la situation (a) on cherche a obtenir la situation (b) .



(a) (b)

On arrive à ce résultat en comptabilisant pour chaque boitier les pattes (qui sont ordonnées) ou une interconnexion "finit", celles ou une interconnexion "commence" et celles ou une interconnexion "continue".

Bien entendu on spécifie au départ que les interconnexions "démarrent" au boitier le plus proche du connecteur.

3) Ordonnancement des colonnes:

Au départ on choisi la colonne dont la longueur totale des segments horizontaux (// à y) servant aux interconnexions entre ses propres boitiers est maximale.

Puis on associe aux autres colonnes une fonction de mérite V_{10}. La colonne ayant la plus grande valeur pour V_{10} sera choisie et placée à droite ou à gauche de la colonne précédente suivant que la valeur de sa fonction V_{11} sera plus petite ou plus grande que la valeur de sa fonction V_{12}.

V_{10} représente le nombre des interconnexions entre la colonne considérée et celle (s) déjà placée (s).

V_{11} (V_{12}) représente la longueur totale des connexions entre la colonne considérée et les colonnes déjà placées, la colonne considérée étant placée à gauche (à droite) des colonnes déjà placées.

IV. 2. 3. Adaptation et implantation de l'algorithme de G. ALIA, G. FROSINI et P. MAESTRINI.

L'implantation se faisant sur un système à micro-processeur nous avons préféré un algorithme du type à construction progressive, plus rapide et aussi plus proprice, nous semble-t-il, à travailler en mode interactif. Nous avons donc choisi comme base de départ

l'algorithme de G. ALIA, G. FROSINI, et P. MAESTRINI, que nous avons quelque peu modifié.

En effet, très souvent, et de plus en plus, la répartition des broches connecteurs est fixée au départ.

Nous pouvons donc considérer le connecteur S comme la juxtaposition d'autant de composants S_j qu'il y aura de colonnes sur la carte.

Dès lors le premier composant de chaque colonne sera le S_j correspondant et les 2e et 3e parties de l'algorithme, c'est a dire l'assignement des boitiers dans les colonnes et l'ordonnancement des colonnes sur la carte, disparaissent.

Cette première partie de la conception de la carte de circuit imprimé se fera néanmoins en deux phases, comme suit:

a) Première phase: introduction du format de la carte, choix et position du connecteur.

Au départ du programme de placement des composants, la machine demande la longueur (axe vertical) et la largeur de la carte. Elle vérifie alors si la surface de la carte est suffisante et en informe l'opérateur. Si la surface prévue s'avère insuffisante elle revient en début de programme. Elle demande ensuite de positionner le connecteur: bord supérieur, inférieur, gauche ou droit de la carte. Puis elle demande encore de préciser de quel côté se trouve la broche n° 1 ainsi que le nombre de broches. Enfin elle demande si le connecteur est un standard; si oui elle demande le nom, si non elle édite la liste des broches et demande de définir les signaux correspondant à chaque broche.

Il lui reste à vérifier s'il y a concordance entre les signaux d'entrée/sortie du schéma et ceux définis pour le connecteur, elle signale les erreurs éventuelles et donne la possibilité de corriger.

Elle termine cette phase de la procédure en visualisant la carte avec la grille de positionnement comme figure ci-dessous:

S_1	S_2	S_3	S_4
01	05	09	0D
02	06	0A	0E
03	07	0B	0F
04	08	0C	10

Elle donne également la liste des composants à placer, et demande à l'opérateur de marquer son accord.

Si l'opérateur marque son accord, la machine passe à la phase suivante de la procédure.

Dans le cas contraire, la machine donne la possibilité à l'opérateur de modifier le format ou/et la position du connecteur. Les résultats de cette modification ayant été présentés et acceptés, la machine passe à la phase suivante.

b) Deuxième phase: procédure de placement des composants.

Cette procédure consiste à appliquer l'algorithme de placement de G. ALIA, G. FROSINI et P. MAESTRINI, modifié comme nous l'avons indiqué précédemment.

Le résultat est visualisé ainsi que le schéma de cablage. La machine demande alors à l'opérateur s'il veut échanger certains composants de colonnes. L'algorithme ayant évalué à chaque fois le lien entre un composant et un groupe de composants, peut avoir placé deux composants fortement liés dans des colonnes différentes.

Ces échanges étant terminés la machine demande à l'opérateur s'il a des échanges à proposer dans l'une ou l'autre des colonnes.

Enfin la machine demande s'il y a des échanges de fonctions a effectuer dans les composants.

Ceci termine la procédure de placement.

l'annexe 4-1 donne les ordinogrammes relatifs à ce qui précède.

Note: la machine pourrait éventuellement vérifier s'il y a deux composants quelconques, de colonnes différentes, qui sont plus fortement liés que le lien moyen dans chaque colonne, mais cela risque d'augmenter considérablement la durée de la procédure.

IV. 3. RECHERCHE DES INTERCONNEXIONS.

IV. 3. 1. Aperçu général du problème:

Ici également aucune des méthodes actuelles ne permet ni d'affirmer au départ qu'il existe une solution, ni de fournir la solution optimale. De même il est impensable d'essayer toutes les solutions possibles: nous aurons donc encore recours à des méthodes heuristiques.

Il existe essentiellement deux familles d'algorithmes de tracé:

La première décompose le plan de la carte en cellules élémentaires et recherche les chemins possibles en explorant systématiquement chaque cellule. Nous présenterons pour cette famille l'algorithme de LEE qui est un des premiers et des plus communément utilisé, nous parlerons également des principaux travaux qui sont venu le compléter.

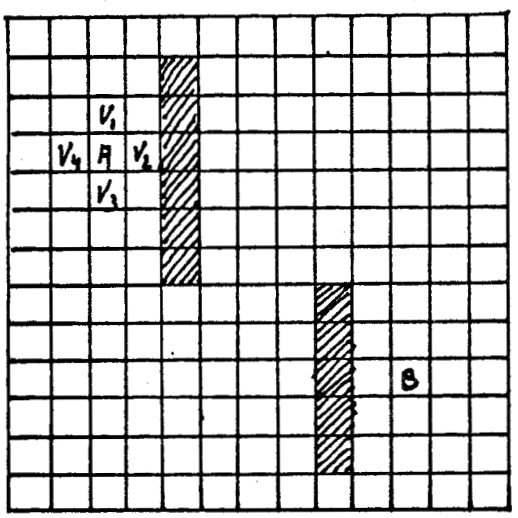
La deuxième famille d'algorithmes travaille directement avec des segments de droites parallèles à un système d'axes rectangulaires qui coïncident généralement avec les cotés de la carte.

Nous présenterons en exemple de cette famille, l'algorithme de MIKAMI et TABUCHI qui nous a semblé particulièrement interessant.

IV. 3. 2. Exemples d'algorithmes.

a) L'algorithme de LEE (1961) (54)

Le plan du circuit est donc divisé par une grille en cellules carrées(voir figure ci-dessous).

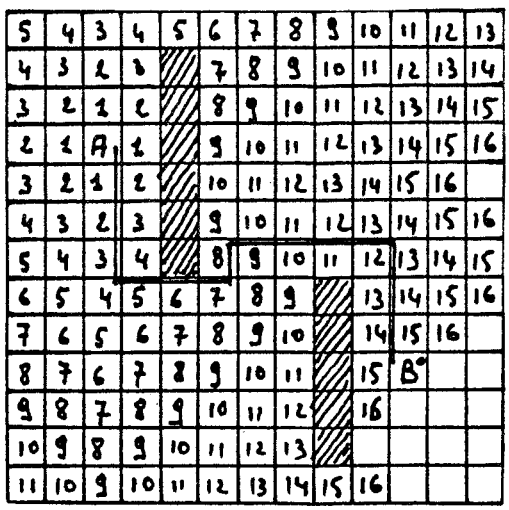


Les cellules sont libres ou non (obstacles).
Soit à rechercher le tracé entre la cellule A et la cellule B.
La technique proposée par LEE est la suivante:

1) on choisi la cellule de départ, soit A.
2) on recherche les cellules voisines de A, soit : V1, V2, V3, V4.

Si ces cellules sont libres on les marque d'un 1. Si elles ne sont pas libres on ne les retient pas. Si aucune des cellules voisines n'est libre, alors il n'y a pas de tracé possible à partir de A. Sinon, on reprend les cellules marquées d'un 1 et on recherche leurs propres cellules voisines libres, que l'on marque d'un 2, et ainsi de suite jusqu'a atteindre B. Si l'on se trouve bloqué (aucunes cellules voisines libres) avant d'atteindre B, c'est qu'il n'y a pas de tracé

possible entre A et B. Sinon il suffit de "remonter", à partir de B , en repérant les cellules voisines marquées n, puis n-1, n-2, ...1, A. (voir figure ci-dessous).



On voit qu'il y a parfois plusieurs chemins possibles et qu'il faut prévoir des critères de choix, par exemple une priorité au non-changement de direction.

Dans l'exemple ci-contre, il y avait un choix à faire au niveaux 14, 9 et 4.

En fait LEE retient tous les chemins possibles et prévoit d'associer à chaque chemin une fonction de coût répondant à un objectif déterminé. Le chemin choisi étant celui ayant la fonction de coût la plus faible.

LEE propose une fonction de coût dans les cas suivants:

- minimiser le nombre de croisements avec d'autres chemins.
- minimiser l'effet de bord.

1) Modification de l'algorithme de Lee par B. AKERS (1967) (3)

AKERS propose de remplacer le marquage des cellules voisines par la séquence 1, 1, 2, 2, 1, 1, 2, 2 ..... en lieu et place de la suite 1, 2, 3, ... n.
Cette modification permet de réduire sensiblement la place mémoire nécessaire.

2) Extension de l'algorithme de Lee aux circuits multicouches par J.M. GEYER. (1971) (16)

Dans l'algorithme de LEE on se contente de marquer chaque cellule analysée et retenue suivant une séquence 1, 2, 3, 4, ...n ou 1, 1, 2, 2, 1, 1, ... GEYER propose de compléter cette information en définissant pour chaque cellule les caractéristiques suivantes, en plus de la simple séquence de marquage:

1) LA : indique pour chaque couche si la cellule est libre ou non (1 bit par couche).

2) LP : indique pour chaque couche si un chemin potentiel a été découvert ou non (1 bit par couche).

3) V : indique si la cellule considérée peut servir à interconnecter les différentes couches (trou métalisé).

Ces définitions permettent par une procédure de marquage et traçage analogue à celle employée par LEE de retrouver les chemins possibles en passant d'une couche à l'autre.

3) L.S. PUGH. (1978) (43)

Suggère de compléter le marquage des cellules par 3 bits :
000 - cellules non marquées
001 - cellule de départ/cible
010 - traçage vers le haut
011 - " " la gauche
100 - " " la droite
101 - " " le bas
110 - cellules non utilisées
111 - limites de recherches.

L'auteur fait encore quelques suggestions permettant de réduire le temps des tracés ou pour éviter les conflits lors du traçage des chemins suivants.

4) Proposition d'une fonction de coût tendant à uniformiser la densité du tracé sur la carte. (1979) (17)

P.A. GRAWAL et M.A. BREUER proposent la fonction de coût suivante:

- pour l'adjonction de la cellule i a un chemin p :

  $C_i = 1 + \alpha D_i$ avec la constante $\alpha \geqslant 0$

- le terme 1 correspond à la contribution au coût en longueur de chemin.

- $D_i = \frac{n}{8}$ ou n est le nombre de cellules occupées parmis les 8 voisines immédiates.

  $0 \leqslant D_i \leqslant 1$

- le coût du chemin p sera alors:

  $$C(p,\alpha) = \sum_{\substack{\text{cellules} \\ i \in p}} C_i = L + \alpha \sum_{\substack{\text{cellules} \\ i \in p}} D_i$$

  L est la longueur du chemin

  $C(p,o) = L$ correspond au coût du chemin de longueur minimale.

Les auteurs indiquent une valeur optimale pour α de l'ordre de 1,25.

b) Algorithme de MIKAMI et TABUCHI. (1969) (37)

Cet algorithme construit le tracé par assemblage de segments de direction X ou Y,

Toutefois chaque face est liée à une direction, soit X, soit Y.

Dès lors trois coordonnées suffisent pour définir un segment:

Xi , Yi , Yj (pour la face Y)

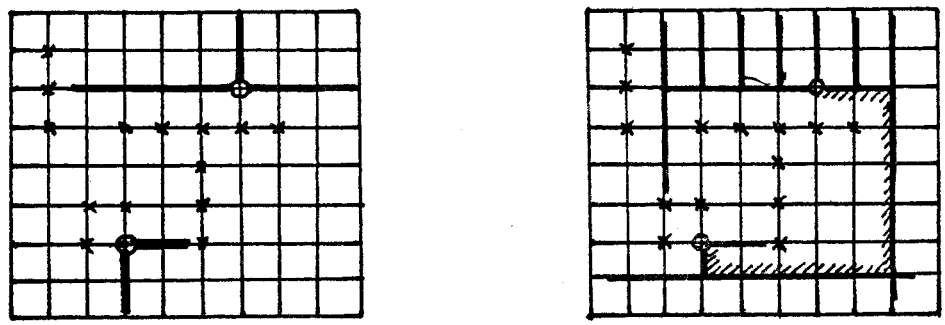
ou Yi , Xi , Xj (pour la face X)

Ce qui précède défini le chemin allant du point Xi , Yi au point Xj , Yj en passant par le point Xi , Yj (2 segments)

La recherche du chemin entre deux points est faite simultanément à partir de ces deux points.

Une solution est obtenue lorsque les deux chemins se rencontrent.

Le dessin suivant explique par lui même et de façon concise l'algorithme :



c) Ordre suivant lequel sont tracées les différentes interconnexions sur la carte.

Un défaut commun aux algorithmes de tracé est de traiter les chemins un à un. Pour éviter que le tracé d'un chemin n'entrave trop celui des suivants, il y a lieu de se poser la question de savoir dans quel ordre il convient de les traiter.

Plusieurs méthodes ont été avancées, il faut toutefois signaler, avant de les présenter, qu'une étude menée par Luther C. ABEL (1), prétend que l'ordre suivant lequel sont traités les chemins à tracer n'a pratiquement aucune influence sur le résulat final: du point de vue nombre de chemins refusés (non trouvés). Notons seulement que cet ordre n'est en fait qu'une façon de "préserver le futur", c'est à dire, chercher à entraver le moins possible les étapes suivantes du traitement.

Généralement les méthodes proposées sont basées sur un classement en fonction de la longueur des connexions.

Souvent on propose de commencer par les connexions les plus courtes, bien que certains prétendent avoir trouvé de meilleurs résultats en commençant par les connexions les plus longues.

En fait comme la longueur, l , d'une connexion est considérée comme étant : $l = \Delta x + \Delta y$
(axe x parralèle au connecteur, coté (généralement) correspondant à la dimension la plus courte de la plaque).

Certaines méthodes donnent priorité aux connexions ayant soit Δx court, soit Δy court.

David W. HIGHTOWER (21) classe les connexions dans l'ordre croissant de $l = \Delta x + 10 \Delta y$.

D'autres méthodes traitent par priorité les connexions dans les régions à fortes densités.

Certaines méthodes sont adaptatives, c'est à dire que l'ordre de traitement est recalculé périodiquement (chaque fois qu'un groupe de n connexions a été traité).

IV. 3. 3. Adaptation et implantation de l'algorithme de LEE.

a) Préliminaires:

Au stade actuel la machine dispose en mémoire, d'une part de la table des interconnexions à réaliser et, d'autre part du format de la carte, de la disposition des composants sur cette carte ainsi que de la liste des composants.

Pour l'exécution de la présente phase du travail nous avons besoin principalement de deux tables:

1 - La table des interconnexions (TIC) telle qu'elle existe déjà.

2 - La table image de la carte (TIM), ou chaque point élémentaire, chaque cellule, sera représenté en mémoire par un octet, donc une adresse mémoire par cellule.

Nous allons rappeler à présent la structure de la table des interconnexions (TIC) puis nous parlerons de la table image (TIM).

1) La table TIC sera structurée de la façon suivante:

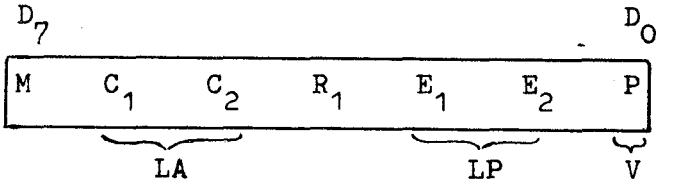
1 - PDI : pointeur de début de table
2 - PFI : pointeur de fin de table
3 - H'FF' sert de frontière entre chaque interconnexion
4 - H'FF' sera suivi de 4 octets donnant successivement et dans l'ordre x_D y_D , x_A , y_A

2) Nous avons dit que la table image TIM contenait un octet par cellule, c'est à dire que chaque adresse mémoire de cette table correspondra à un point de la carte.

1° Structure de TIM.

En fait cette table doit être structurée de façon à permettre la marquage des cellules et la recherche des chemins selon l'algorithme de LEE, puis à visualiser simplement le dessin de la carte avec les interconnexions trouvées.

C'est pour pouvoir faire le marquage et la recherche des chemins selon l'algorithme de LEE que chaque cellule sera représentée en mémoire par un octet. Cet octet sera codé de la manière suivante; similaire à la façon proposée par J.M. GEYER:



M : indique si la cellule est occupée (0) ou non (1) par un mur

C_1 : libre (1) ou non (0) sur la couche 1

C_2 : libre (1) ou non (0) sur la couche 2

R_1 : } utilisée lors de la phase de marquage

R_2 : }

'00' : extrémité de chemin

'11' : non encore marqué

'01' } : marquage suivant la séquence

'10' } 1, 1, 2, 2, 1, 1, 2, 2, ...

E_1 : chemin potentiel trouvé sur la couche 1 → '0'

E_2 : chemin potentiel trouvé sur la couche 2 → '0'

P : passage métalisé : '0' (oui)

'1' (non)

2° Initialisation de TIM.

Au début de cette partie du travail il va donc falloir initialiser la table TIM.

On localisera d'abord cette table à partir d'un pointeur de début de table (PDM) et du format de la carte (x,y) rentré précédemnt.

Pointeur de table, PFM = PDM + x.y.

La localisation de la table étant ainsi faite il faut initialiser le contenu de chacun de ses octets:

1 - Tous les octets de la table seront mis à la valeur H'FF' Ce qui convient pour l'initialisation de tous les bits R_1, R_2, E_1, E_2, de tous les octets.

2 - M_1, C_1, C_2, P, seront ensuite initialisés individuellement et pour chacun des octets de la façon suivante:

- M_1, C_1, C_2, P des octets correspondant aux broches des composants, ainsi que du connecteur seront mis à 'O' automatiquement par la machine.
  Pour cela la machine se servira de la même procédure que pour établir la table des interconnexions.
  Elle se sert de la liste des composants et de la bibliothèque "Matériel" pour obtenir la position exacte des broches des composants par rapport à leur centre . La position du centre de chaque composant étant connue à la fin de la phase du placement le calcul des coordonnées de chacune des broches sur la carte est simple.
  Pour retrouver dans la table TIM l'adresse $A_{x,y}$ de la cellule de coordonnée x,y sur la carte il suffit du calcul suivant:

$$A_{x,y} = PDM + N_y + x$$

N = X max = x rentré pour le format de la carte.

- Il devra être possible à l'opérateur d'imposer des "murs" supplémentaires éventuels sur la carte. Pour cela le résultat de la phase 'a' sera visualisé et l'opérateur aura alors la possibilité de rentrer les "murs" désirés.

L'annexe 4-2 donne l'ordinogramme de ce qui vient d'être décrit.

b) Recherche des voisins et marquage:

Tout étant prêt nous pouvons à présent demander à la machine d'appliquer la première partie de l'algorithme de LEE.

Elle se servira de la table des interconnexions (TIC) et de la table des images (TIM) ainsi que d'une table contenant la suite des cellules dont on doit encore examiner les voisins (TCV).

La table TCV sera structurée de la même façon que (TIC), avec un pointeur de début, PDV, et un pointeur de fin incrémentable, PFV. Les cellules seront mémorisées par leur coordonnées x,y dans l'ordre.

Il faudra encore établir:

I' = compteur de voisins examinés pour une cellule $1 \leqslant I' \leqslant 4$

I" = compteur de voisins examinés par génération

N_G = nombre de cellules par génération

$N_G^+ = N_G + 4$

("onde carrée" qui en grandissant donne par génération, pour chaque cellule de la génération précédente, une cellule de la génération nouvelle, sauf deux des cellules de "pointe" qui en donnent trois).

G = numéro de la génération

M = nombre de cellules rejetées par génération (mur ou hors cadre)

Notons que nous ne suivons pas exactement ni le codage, ni le marquage proposé par J.M. GEYER:

P n'est pas identique à V

E_1 E_2 ne sont pas marqués exactement comme LP

P = O uniquement s'il y a passage métallisé

V donne les passages possibles

E_1 E_2 tient compte d'un chemin réellement possible sur l'une ou l'autre face.

Avec LP il faut mémoriser toutes les cellules antérieures depuis le dernier V possible.

L'ordinogramme correspondant est donné dans l'annexe 4-3

c) Recherche des chemins possibles et choix:

Nous abordons la dernière phase du travail avant la visualisation du résultat.

Pour cette phase la machine utilisera encore la table image (TIM) et se servira en plus des tables suivantes:

- TEP : table d'embranchements possibles. Contient les coordonnées des cellules du chemin que l'on construit et d'où pourrait partir un embranchement.

- TCC : table du chemin en construction. Contient les coordonnées des cellules du chemin que l'on construit.

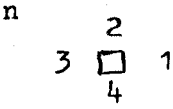
- TCR : table du chemin de référence. Contient les coordonnées des cellules du meilleur chemin trouvé jusqu'à présent et la couche sur laquelle se trouve ce chemin, ainsi que sa longueur et le nombre de changements de direction qu'il comporte.

Nous aurons encore les paramètres suivants:

x_A, y_A = coordonnées du point d'arrivée.

D = nombre de changements de direction

F = numéro du voisin: $1 \leqslant F \leqslant 4$



F_0 = mémorisation de la direction prise à l'étape précédente

VF = voisin n° F

Les tables TCC, TCR, et TEP ont une structure analogue aux tables TIC et TCV.

Pour TCC nous aurons les pointeurs PDC et PFC

I = longueur du chemin en construction

D = nombre de changements de direction

Pour TCR nous aurons les pointeurs PDR et PFR

I_r = longueur du chemin de référence

D_r = nombre de changements de direction

Pour TEP nous aurons les pointeurs PDE et PFE

L'ordinogramme correspondant est donné dans l'annexe 4-4

IV. 4. CONCLUSION.

Nous avons à présent terminé l'étude de la carte imprimée, donc le travail que nous proposions de faire. L'image de cette carte est contenue dans la table TIM. Pour la visualisation il suffit de lire les bits M, C_1 et C_2 de TIM et de transcrire l'information qu'ils contiennent dans la zone mémoire image de l'écran. Notons que ces mêmes informations, ainsi que celle contenue dans le bit P, peuvent être exploitées directement pour la réalisation de la carte de circuit imprimé: dessins des films, perçage, etc.....

CONCLUSIONS GENERALES.

Nous avons donc pu mettre sur pied l'outil que nous envisagions au départ et cela sans problèmes majeurs. Bien sur nous n'avons pu réaliser qu'une étude générale du sujet et beaucoup reste à faire.

Pour affiner cet outil il nous semble qu'il y aurait lieu avant tout de développer les banques de données, tant celle relative au premier chapitre, c'est à dire les modules fonctionnels, que celle correspondant aux composants. En effet au stade actuel la pratique du système serait sans doute encore le meilleur guide permettant d'orienter efficacement l'aide à apporter à l'établissement de la description fonctionnelle, en montrant comment présenter les informations données par la machine au concepteur. De même seul un choix entre un nombre de composants suffisants permettrait de juger valablement de l'interêt de la méthode de choix proposée à ce niveau. Notons à cet effet que le volume de cette dernière banque de données, ainsi que la fréquence à laquelle on devra normalement y accéder, incitera fort probablement très rapidement à échanger l'enregistreur à cassettes contre une unité à disques.

Au niveau de la simulation il y aurait lieu de compléter les vérifications par la machine qui tendent à rechercher les erreurs grossières au niveau de la description fonctionnelle. Nous n'avons pas vérifié, par exemple, si un "chemin-signal" aboutit effectivement. Nous n'avons pas non plus vérifié si toutes les sorties ou entrées sont effectivement connectées.

Pour ce qui concerne la simulation il conviendrait d'approfondir l'évaluation du schéma matériel.

Au sujet de la conception de la carte de circuit imprimé l'expérience devra dire s'il faut ou non apporter une aide au concepteur

sur le choix de la place des fonctions dans les composants. On pourrait aussi nous reprocher d'avoir choisi pour la recherche des interconnexions l'algorithme de LEE alors que nous nous proposons de travailler avec un petit système. L'algorithme de LEE utilise effectivement beaucoup de place mémoire relativement à d'autres algorithmes, mais la taille des cartes imprimées que nous envisageons (format "Europa" simple ou double) permet de ne pas avoir de problèmes pour ce point particulier.
D'autre part l'algorithme de LEE est certainement un des plus efficace. Il serait peut-être utile toutefois de le doubler d'un algorithme plus rapide et nécessitant moins de zone mémoire et de n'y faire appel que dans un deuxième temps pour la recherche des interconnexions refusées par le premier.

Nous terminerons ce travail en soulignant que l'évolution de la technologie nous conduit de plus en plus, pour les problèmes de logiques, vers les solutions programmées. Aussi nous croyons que c'est essentiellement dans cette direction que doivent désormais porter les efforts d'aide à la conception.

ANNEXE 1.1 : Système GENIUS à microprocesseur 2650 de signetics.

1 - Fiche signalétique du microprocesseur 2650

1° Caractéristique de base:

- C'est un microprocesseur 8 bits
- Alimentation + 5 Volts uniquement
- 525 mW de consommation maximum
- Horloge monophase
- Opérations statiques: pas de fréquence d'horloge minimum
- Durée d'un cycle instruction: 3 à 6 µsec. avec une fréquence d'horloge de 2 MHZ (max).

2° Interfaçage:

- Entrées, sorties, compatibles TTL.
- Bus "3-états", accès mémoire direct possible
- Interfaçage mémoire et entrées-sorties asynchrones par rapport à l'horloge du microprocesseur.
- Possibilité de 256 entrées-sorties 8 bits parallèles
- Possibilité d' 1 entrée/sortie série au niveau du boitier.

3° Architecture:

- Bus de données 8 bits, bidirectionnel et "3-états"
- 15 lignes de bus d'adresses, soit 32 K octets de mémoire adressable.
- Structure interne en 8 bits parallèles.
- Sept registres 8 bits internes (adressables)
- Pile d'adresses de retour de sous-routine, à 8 niveaux, interne.
- Mot d'état programme sur deux octets.

- Un seul niveau d'interruption vectorisé
- Les sous-routines d'interruption peuvent être localisées n'importe où dans la mémoire.
- Additionneur séparé pour le calcul des adresses de données.

4° Jeu d'instructions:

- 75 instructions
- Types d'instructions classiques:
  - arithmétique
  - logique
  - sauts conditionnels et inconditionnels
  - branchement vers les sous-routines
  - entrées-sorties
- Instructions particulières:
  - entrées-sorties, deux portes directement décodées sur le boitier.
  - instruction de masquage immédiat
- 8 modes d'adressages différents.

2 - Le système " GENIUS "

1° Le matériel:

Le système sur lequel l'étude est réalisé s'appelle " GENIUS " et consiste en 8 cartes de bases:

1 - Carte CPU
2 - Carte RAM 4K
3 - Carte PROM 7K
4 - Carte TIMER
5 - Carte d'interruption
6 - Carte interface TRC + cassette.
7 - Carte de programmation pour EPROM 2708 et 2716
8 - Carte de fond.

Il est possible d'inclure plusieurs cartes RAM et cartes PROM (Carte PROM "expenssion" de 8K).

Le système est actuellement complété d'une console de visualisation et d'un enregistreur à cassettes.

2° Le logiciel:

Du point de vue logiciel le système comporte:

- un assembleur 4K
- un basic 8K
- un moniteur 7K

Le moniteur comprend:

- un désassembleur
- les commandes:
  - A : visualisation et altération du contenu mémoire
  - B : points d'arret
  - C : levée de points d'arret
  - D : déplacement de blocs de données
  - F : poursuite du programme après un point d'arret
  - G : exécution du programme à l'adresse -
  - L : liste du contenu mémoire
  - S : vérification et modification des registres internes
  - V : programmation de PROM
  - R : lecture de cassette
  - W : écriture sur cassette
- l'accès aux sous-routines suivantes:
  - effacement de l'écran
  - écriture sur l'écran

- sauvetage du contenu des registres internes
- remise en place du contenu des registres internes
- conversion hexadécimal / ASCII
- conversion ASCII / hexadécimal
- déplacement de blocs de données en mémoire
- déplacement du curseur
- lecture de cassette
- écriture sur cassette.

Notons encore que la capacité de mémorisation sur une piste de cassette est de l'ordre de 300 K octets et que l'on peut y mémoriser des blocs de longueurs variables (de 1 octet à 64 K octets).

ANNEXE 1.2. Ordinogramme de l'Editeur

Messages et remarques:

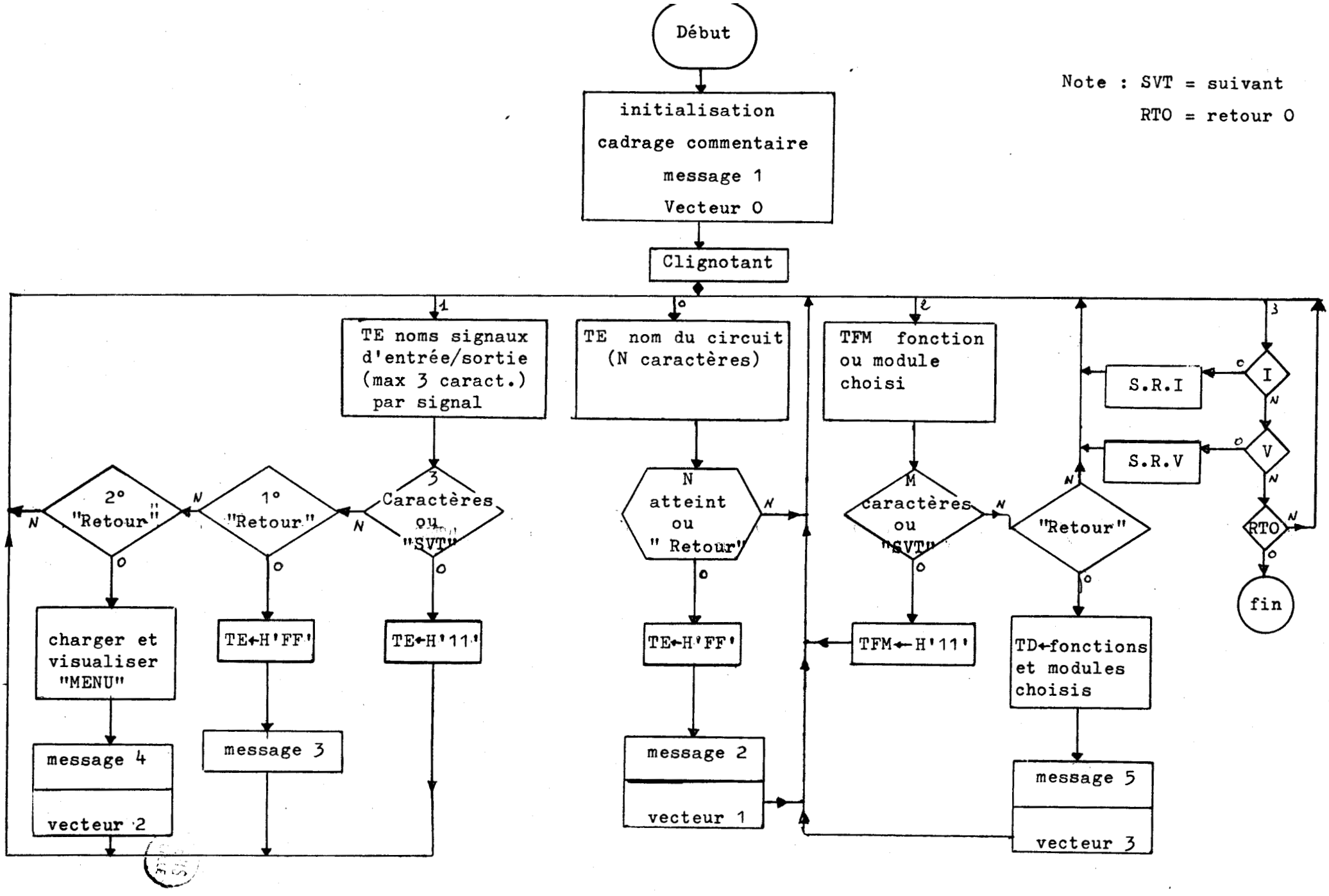
1 - Remarques:

S.R.I sous-routine d'introduction de description (édition), elle permet aussi un retour direct en "Vect.3" ou en "Vect.2".

S.R.V sous-routine de visualisation des descriptions des fonctions ou modules choisis. Possibilité de retour en "Vect.3" ou en "Vect.2".

2 - Messages:

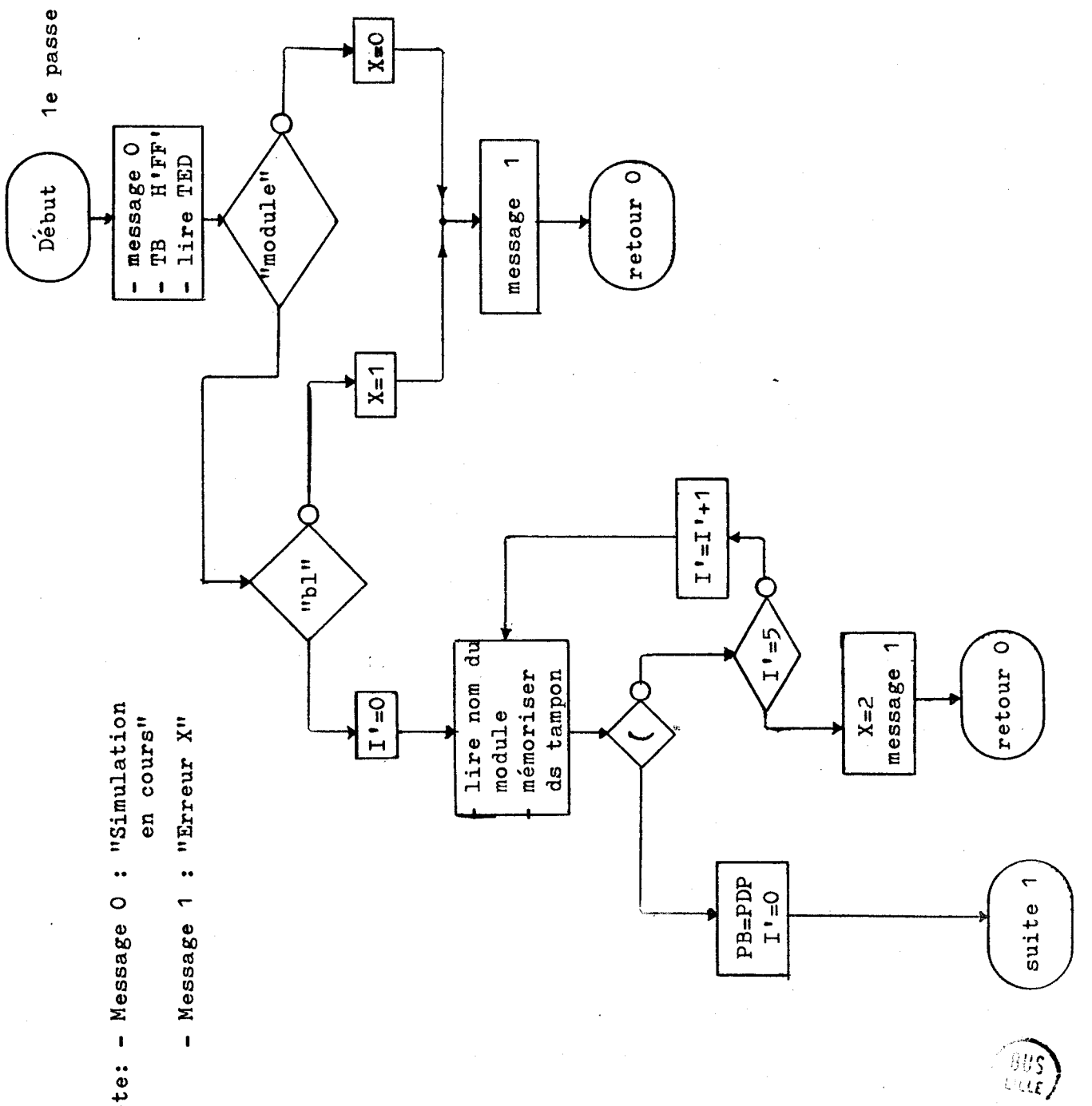
Message 1 : Quel est le nom du circuit ?
Message 2 : Rentrez les noms des signaux d'entrée :
Message 3 : Rentrez les noms des signaux de sortie :
Message 4 : Choisissez les fonctions et modules qui vous intéressent :
Message 5 : Vous pouvez visualiser les descriptions (V) ou éditer la vôtre (I)

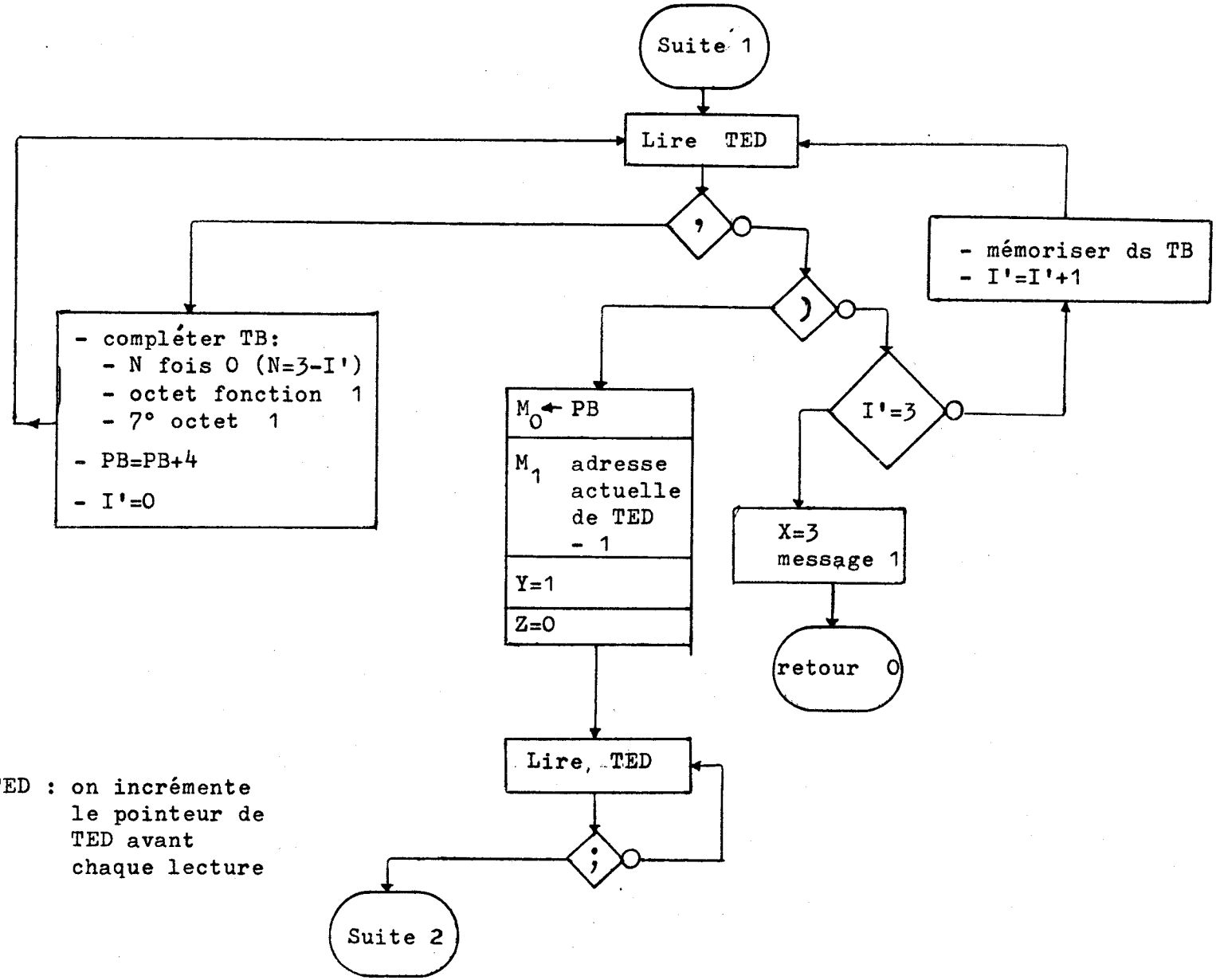


Note : SVT = suivant
RTO = retour 0

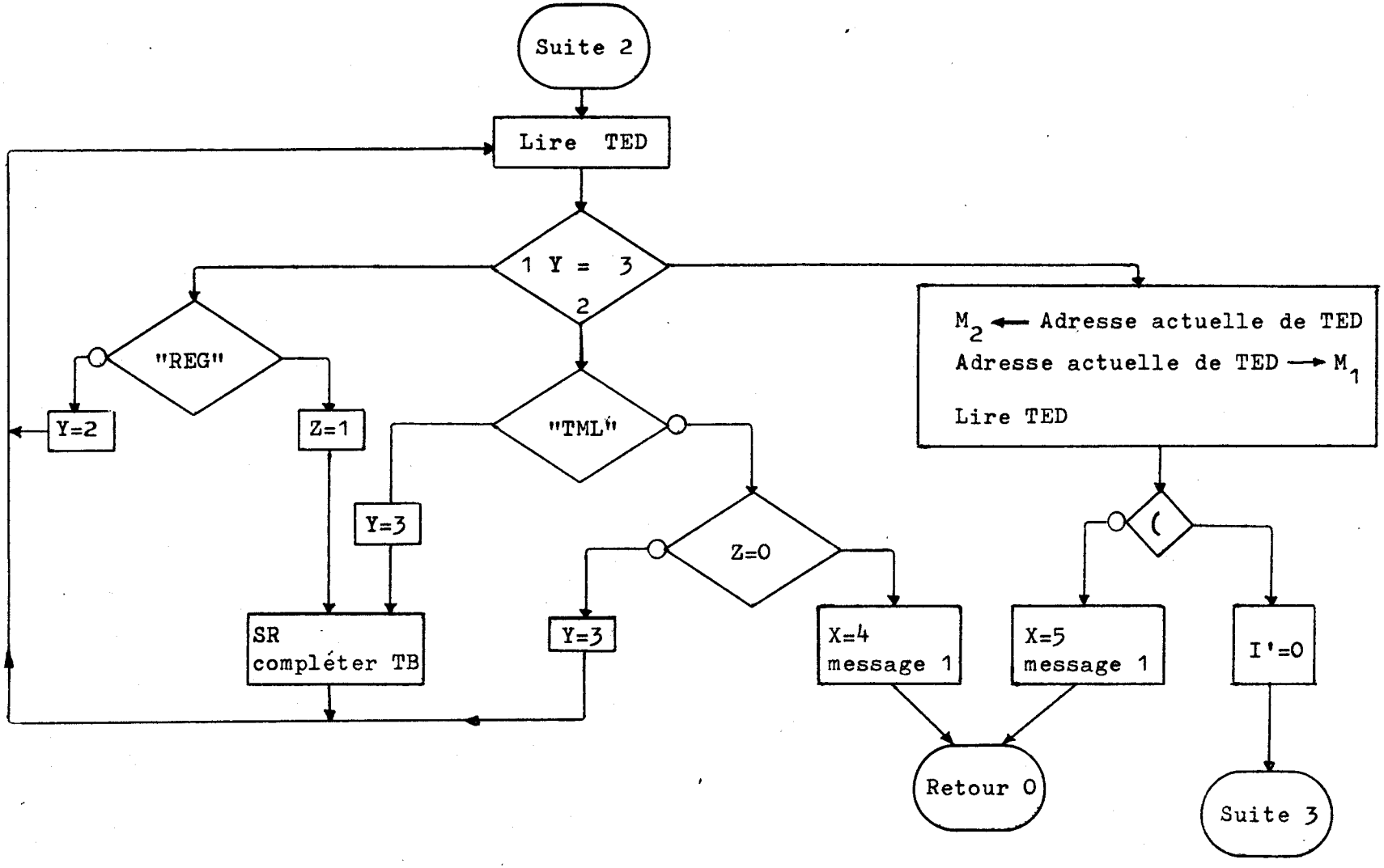
ANNEXE 2.1 Création de la table de base: (ordinogramme)

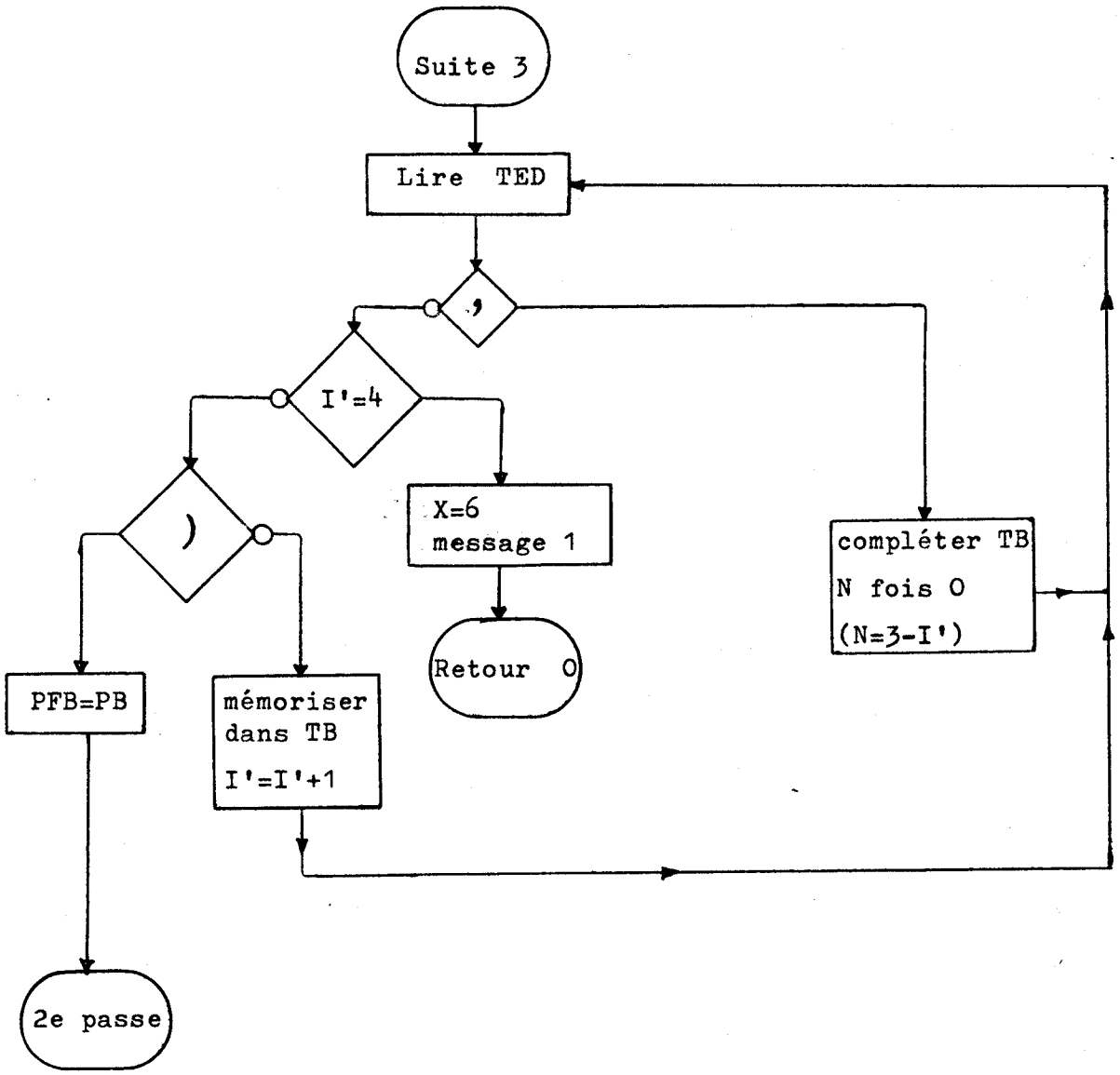
a) Première passe



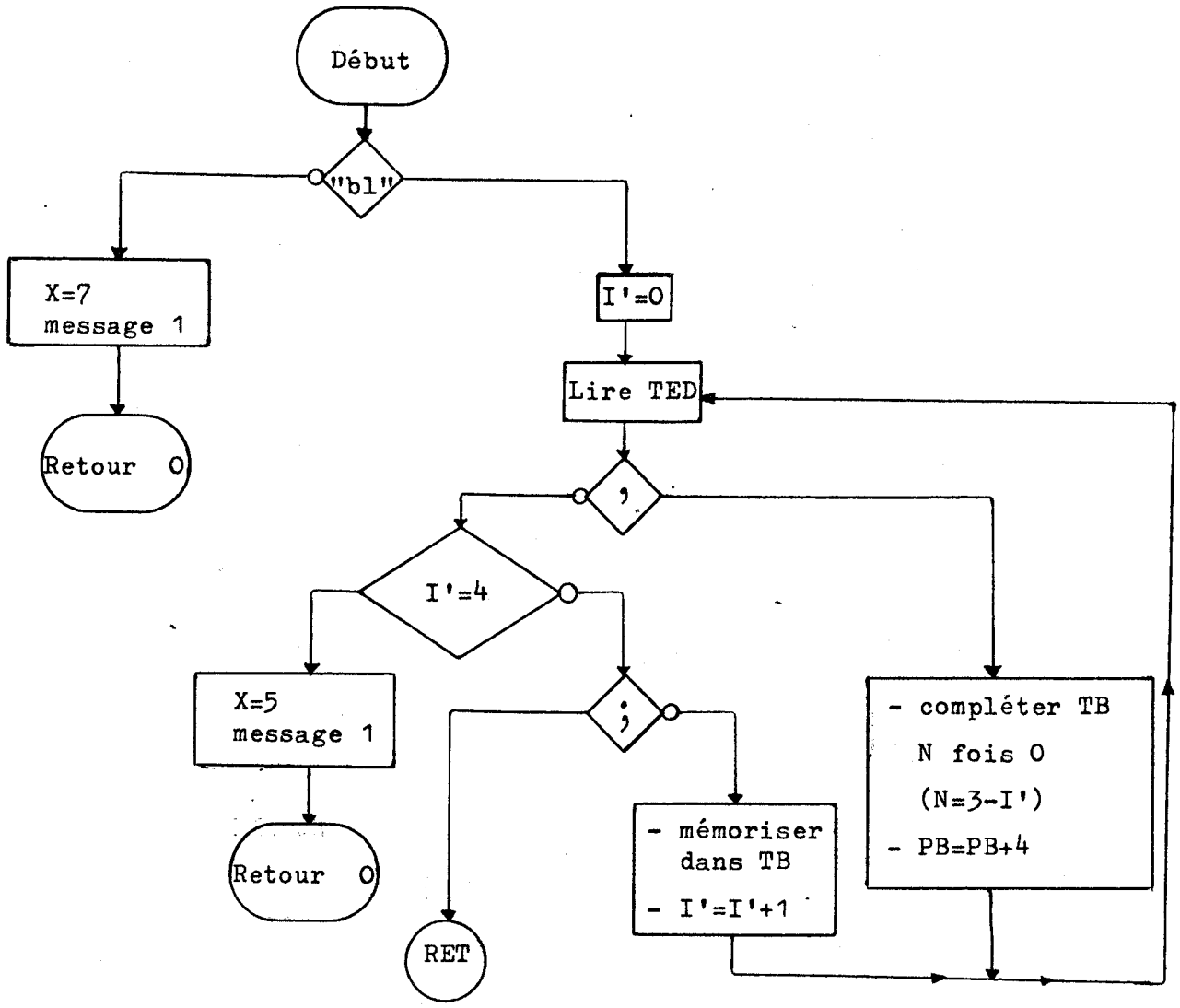


Note: Lire TED : on incrémente le pointeur de TED avant chaque lecture

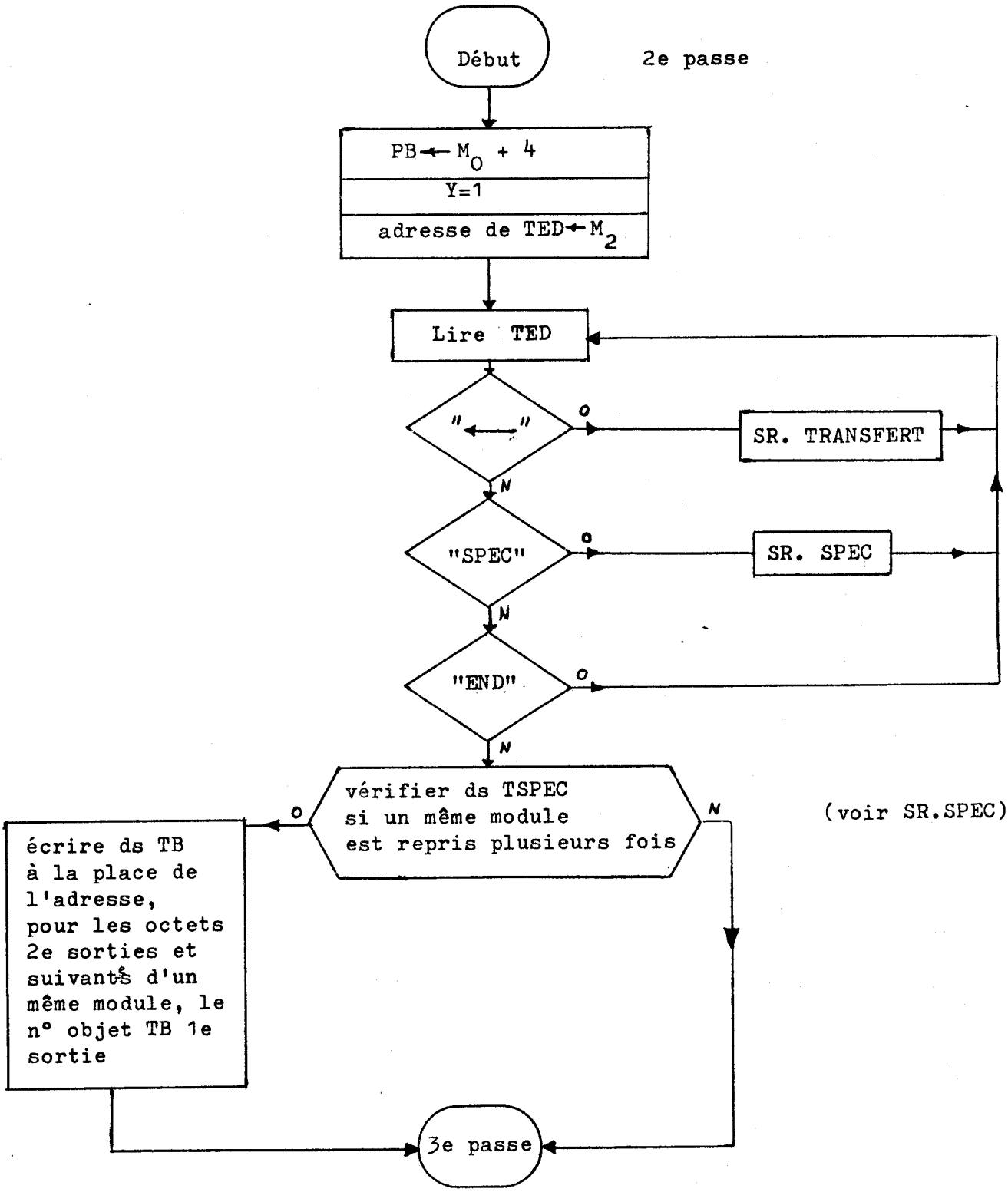

Suite 2
Lire TED
1 Y = 3
2
"REG"
Y=2
Z=1
"TML"
Y=3
SR
compléter TB
Y=3
Z=0
X=4
message 1
M2 ← Adresse actuelle de TED
Adresse actuelle de TED → M1
Lire TED
(
X=5
message 1
I'=0
Retour 0
Suite 3


Suite 3
Lire TED
,
I'=4
)
X=6
message 1
Retour 0
compléter TB
N fois 0
(N=3-I')
PFB=PB
mémoriser
dans TB
I'=I'+1
2e passe

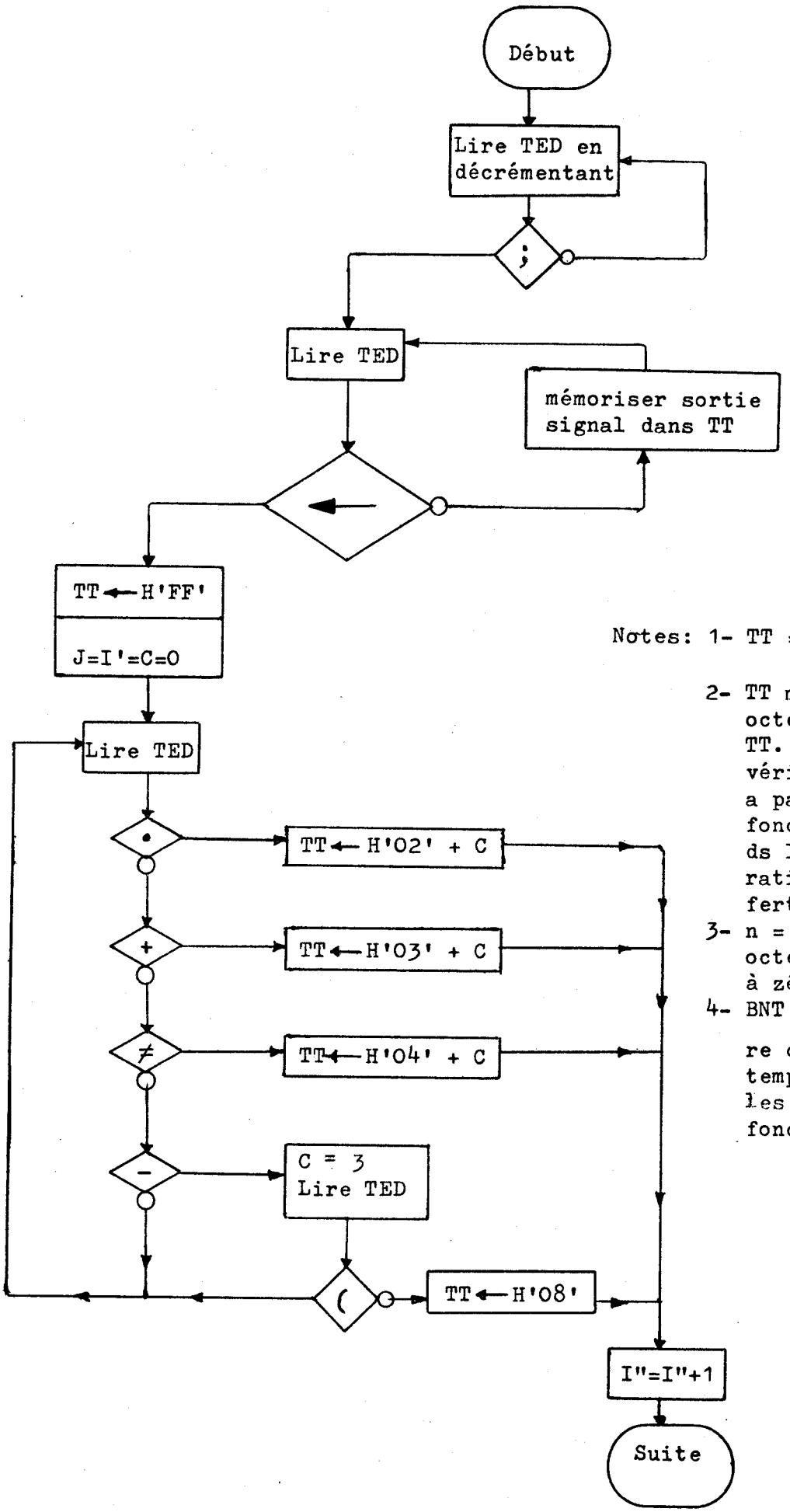
S.R. "Compléter TB"



b) Deuxième passe



S.R. Transfert

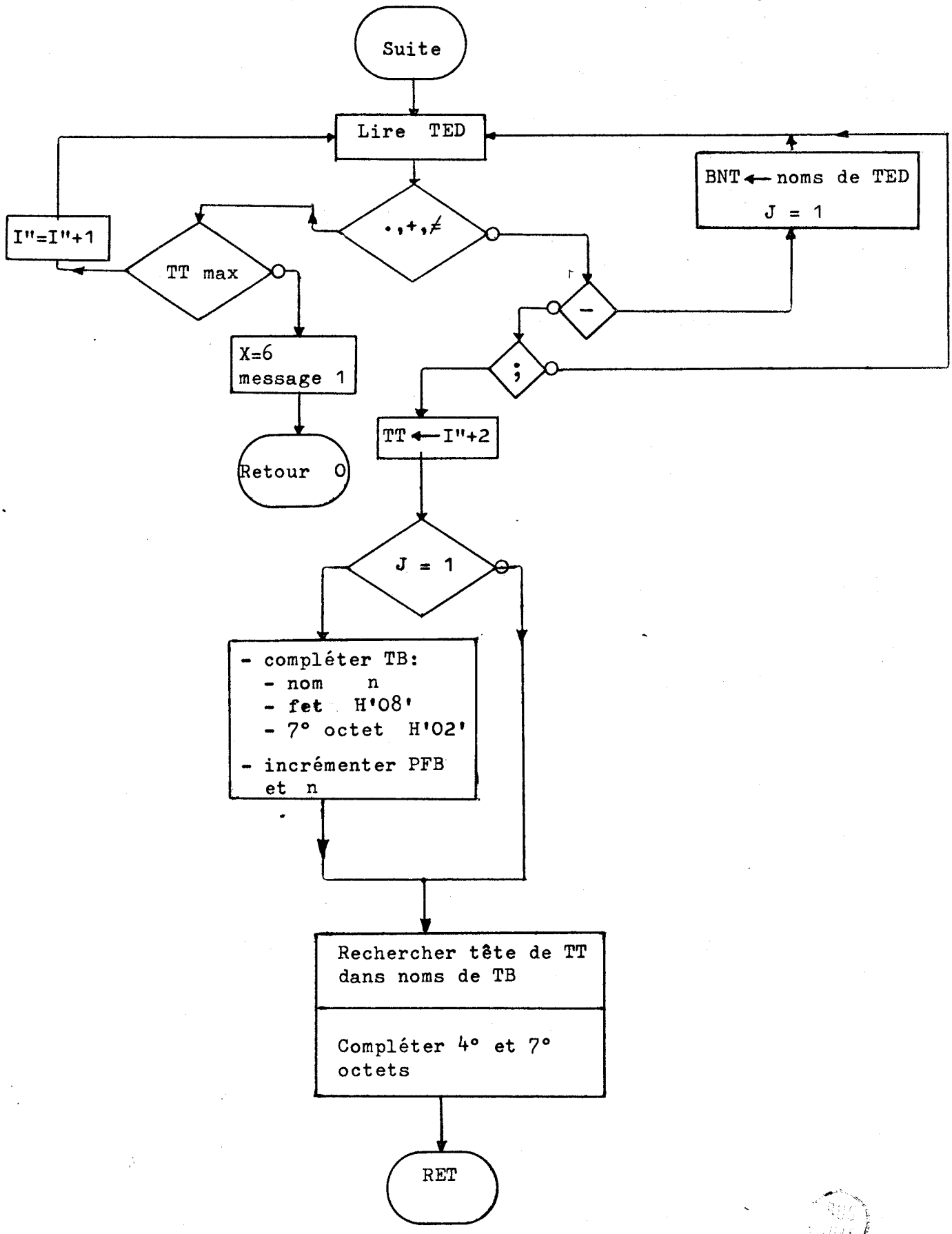


Notes: 1- TT = table temporaire

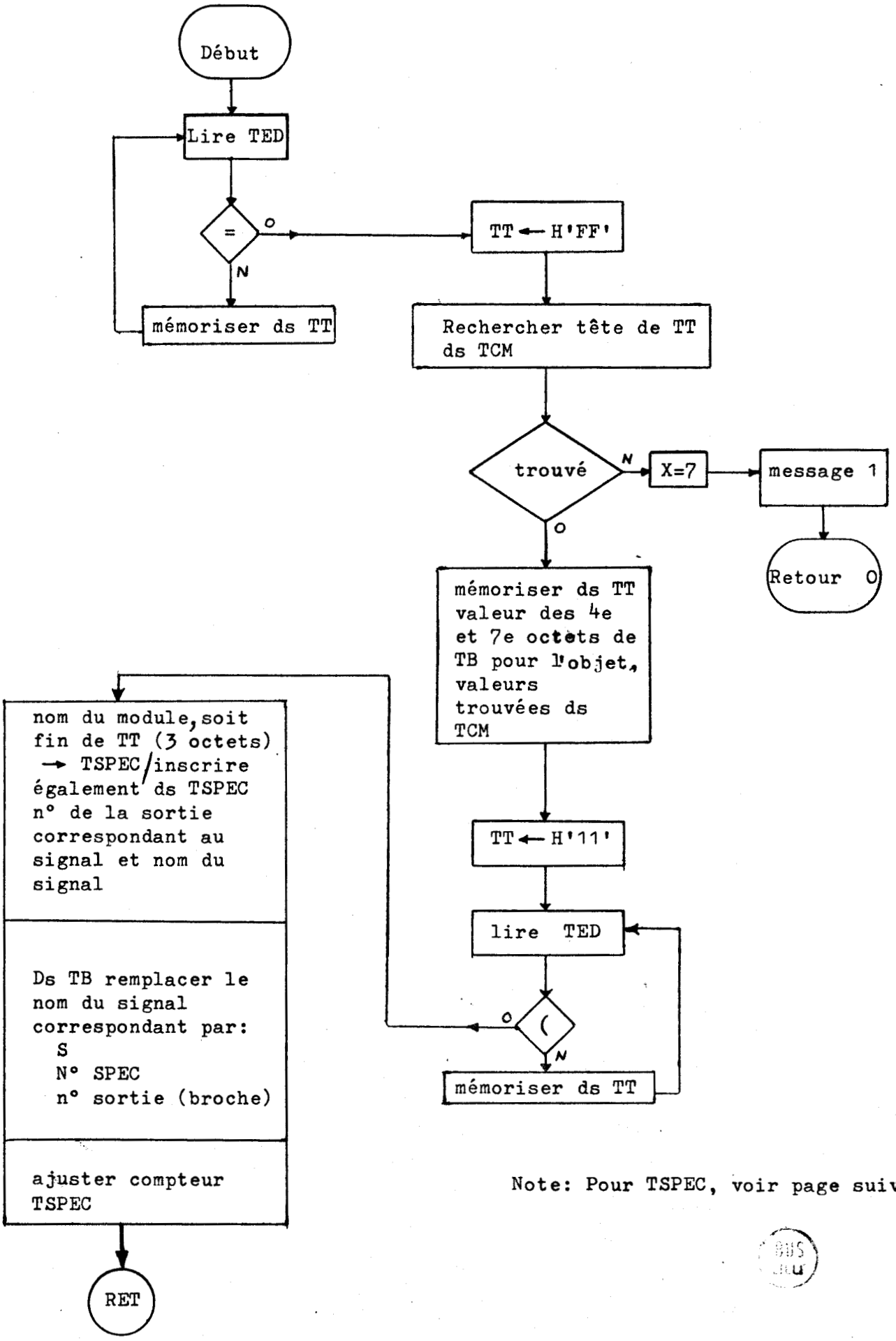
2- TT max = dernier octet rentré ds TT. Permettra de vérifier s'il n'y a pas plusieurs fonctions différent ds la même déclaration de transfert.

3- n = compteur 2 octets initialisés à zéro

4- BNT = zone mémoire contenant temporairement les sources des fonctions n° 8


Suite
Lire TED
BNT ← noms de TED
J = 1
.,+,≠
I"=I"+1
TT max
X=6
message 1
Retour 0
-
;
TT ← I"+2
J = 1
- compléter TB:
- nom n
- fet H'08'
- 7° octet H'02'
- incrémenter PFB et n
Rechercher tête de TT dans noms de TB
Compléter 4° et 7° octets
RET

S.R. SPEC



Note: Pour TSPEC, voir page suivante.

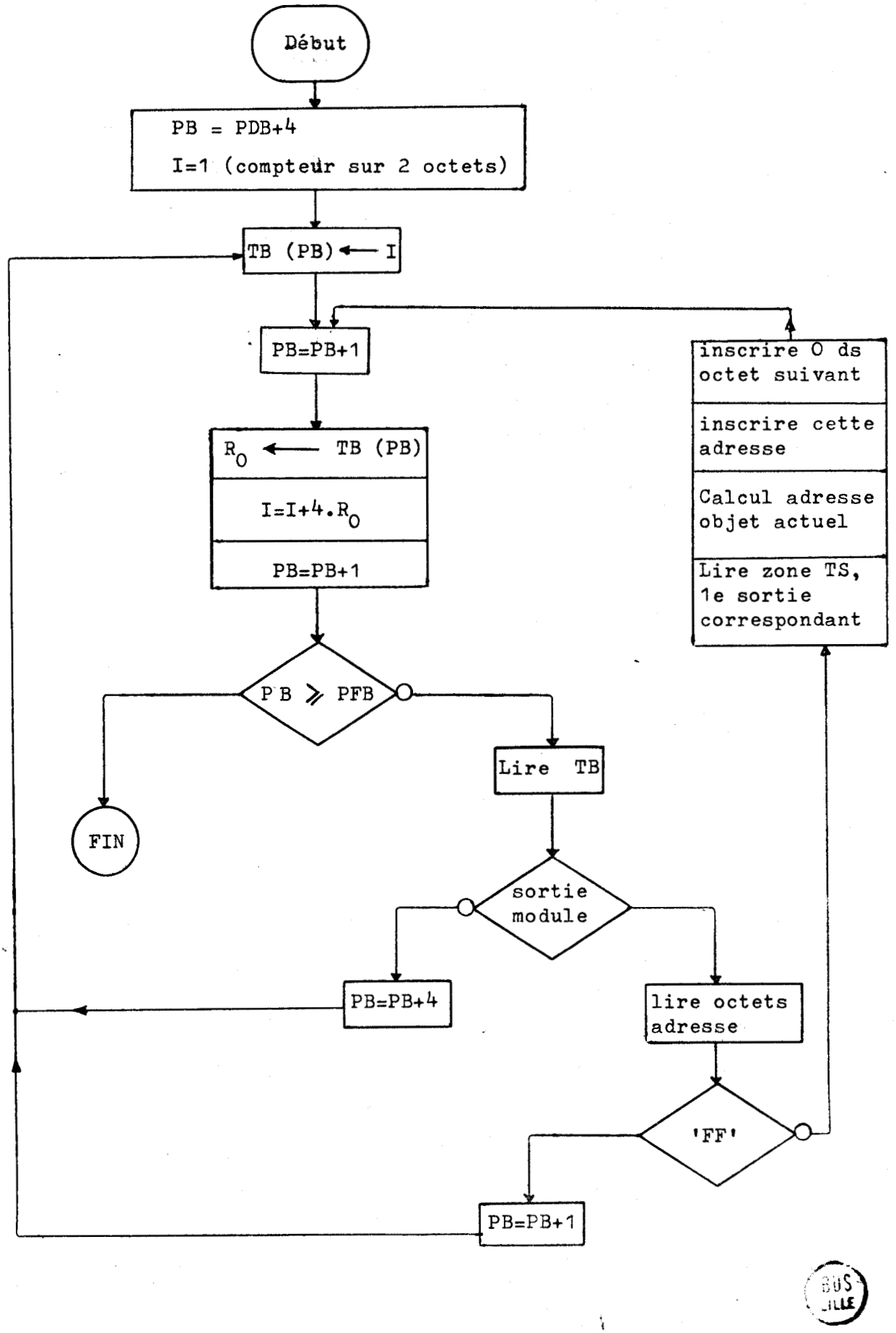
Note: TSPEC table permettant de mémoriser les signaux qui sont affectés à chacune des sorties des modules utilisés.
Chaque objet comprend 7 octets
- 3 pour le nom du module
- 1 pour le n° de sa sortie
- 3 pour le nom du signal correspondant.

c) Troisième passe:

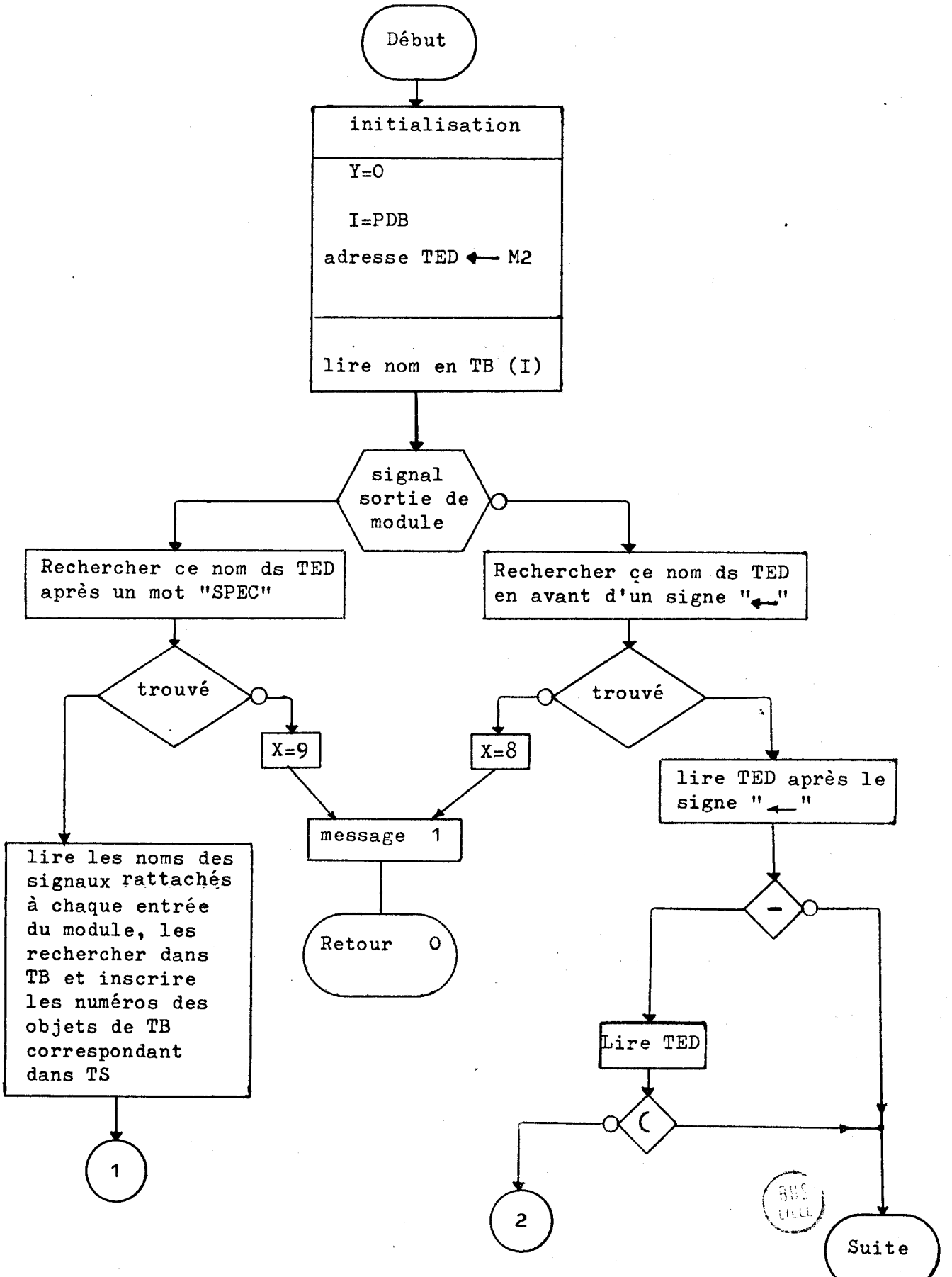
Remarque préliminaire sur la signification de 'FF':

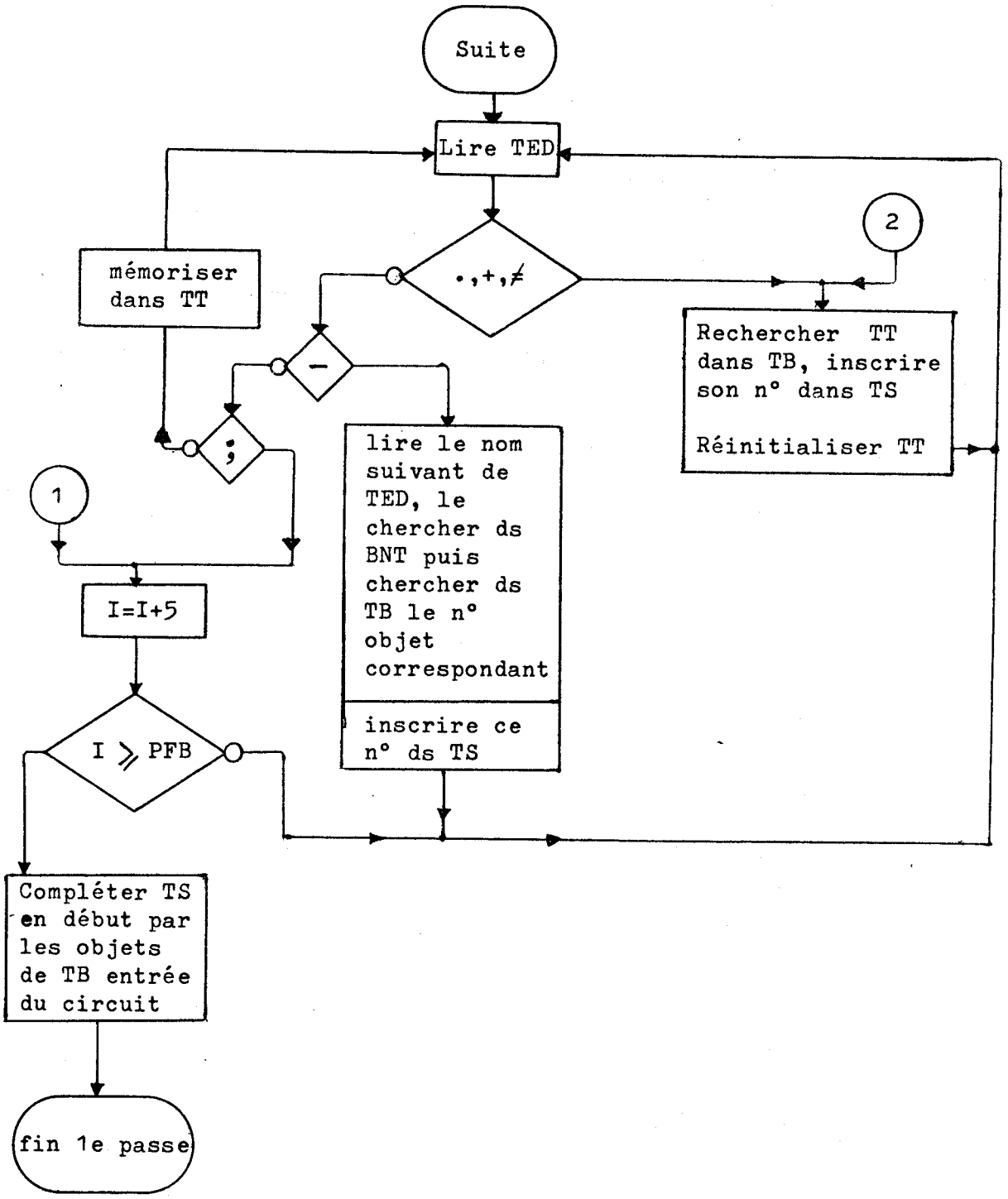
A l'initialisation les octets d'adresse (5° et 6°) sont chargés par 'FF'. On détecte la présence d'une 1e sortie d'un module par 'FF'. On remplit la zone d'adresse de TS normalement. Pour les sorties suivantes les octets d'adresse ne contiendront pas 'FF', mais bien le n° objet TB de la 1e sortie.

L'ordinogramme de la troisième passe est donné page suivante.


Début
PB = PDB+4
I=1 (compteur sur 2 octets)
TB (PB) ← I
PB=PB+1
R0 ← TB (PB)
I=I+4.R0
PB=PB+1
PB ⩾ PFB
FIN
Lire TB
sortie module
PB=PB+4
lire octets adresse
'FF'
PB=PB+1
inscrire 0 ds octet suivant
inscrire cette adresse
Calcul adresse objet actuel
Lire zone TS, 1e sortie correspondant

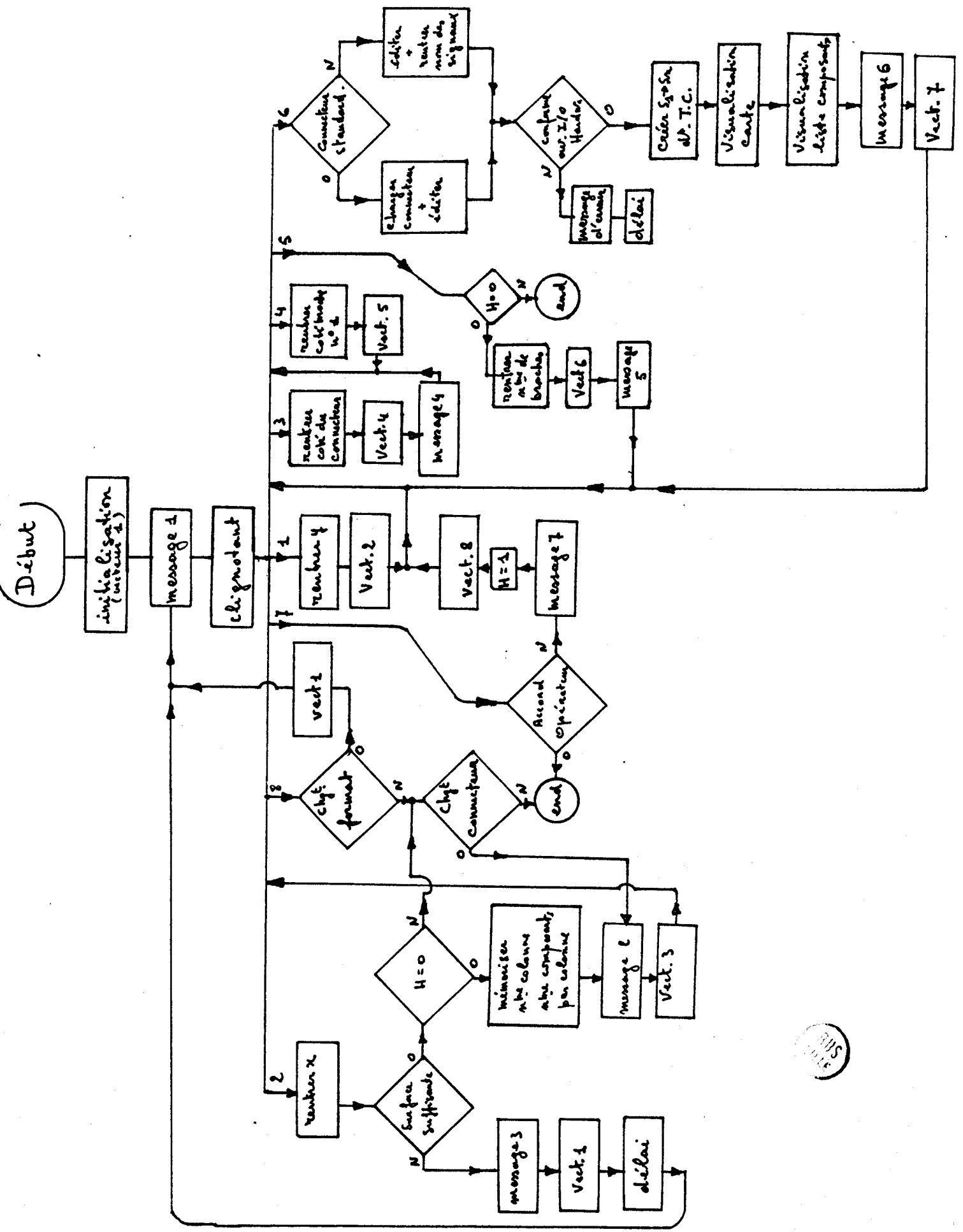
ANNEXE 2.2 Organigramme de création de TS




Suite
Lire TED
mémoriser dans TT
•,+,≠
2
Rechercher TT dans TB, inscrire son n° dans TS
Réinitialiser TT
−
;
lire le nom suivant de TED, le chercher ds BNT puis chercher ds TB le n° objet correspondant
inscrire ce n° ds TS
1
I=I+5
I ⩾ PFB
Compléter TS en début par les objets de TB entrée du circuit
fin 1e passe

ANNEXE 4-1.

a) Ordinogramme de la première partie de la procédure:



b) Messages de la première partie de la procédure:

1) Message 1 : - Donnez la longueur de la carte : ' - '
- Donnez la largeur de la carte :

2) Message 2 : Le connecteur sera-t-il placé
- sur le bord supérieur ? '-'
- sur le bord inférieur ?
- sur le bord gauche ?
- sur le bord droit ?

3) Message 3 : Format incompatible avec le nombre de composants.

4) Message 4 : La broche n°1 du connecteur sera
- du coté gauche : '-'
- du coté supérieur : (sera sauté si la réponse est oui à la ligne précédente.)
- nombre de broches ?

5) Message 5 : Le connecteur est-il standard ? '-'
Quel est son nom ?

6) Message 6 : Etes-vous d'accord avec la disposition de la carte ? '-'

7) Message 7 : - change-t-on la forme ? '-'
- change-t-on le connecteur ?

c) Deuxième partie de la procédure

Nous travaillons ici sur deux tables:

1) La table des connexions (TC) créée en fin de la 3ème partie
(transposition en schéma Matériel)

Structure de TC :

a) PDC : pointeur de début de table TC

b) PFC : pointeur de fin de table TC

c) chaque nouveau composant dans la table est précédé d'un octet contenant la valeur H'11'

Notons que H'11' sert en fait de frontière entre deux composants décrits dans la table TC

d) H'11' est suivi du code ASC II du composant dont la description des connexions broche par broche suit.
Si l'on désire retrouver le type de composant correspondant il faut se référer à la table des composants (TCO) également créée dans la 3ème partie de ce travail.

e) H'FF' suivra le code ASC II précédent et se représentera de manière à servir de frontière entre les descriptions des connexions de chacune des broches.

f) H'FF' sera suivi d'un octet contenant le n° de la broche dont on va donner les connexions .

g) ce numéro sera suivi d'autant de paires d'octets qu'il y a de connexions partant de la broche considérée.
Le premier octet de cette paire donne le code ASC II du composant vers lequel part la connexion décrite. Le deuxième octet spécifie le n° de la broche ou arrive la connexion.

2) La table des composants sélectionnés (TCS)

Structure de TCS :

a) PDS : pointeur de début de table TCS

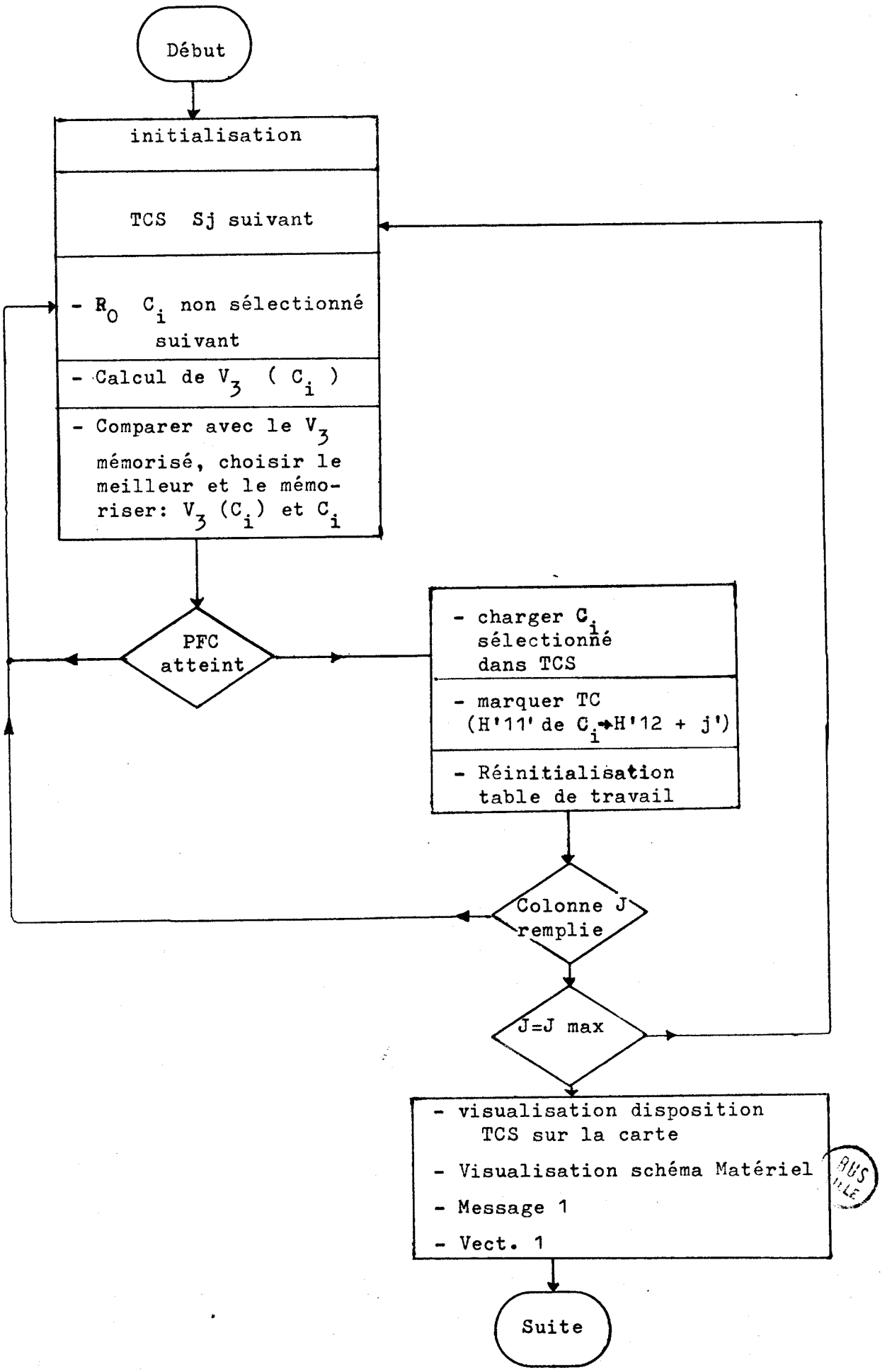
b) PFS : pointeur de fin de table TCS

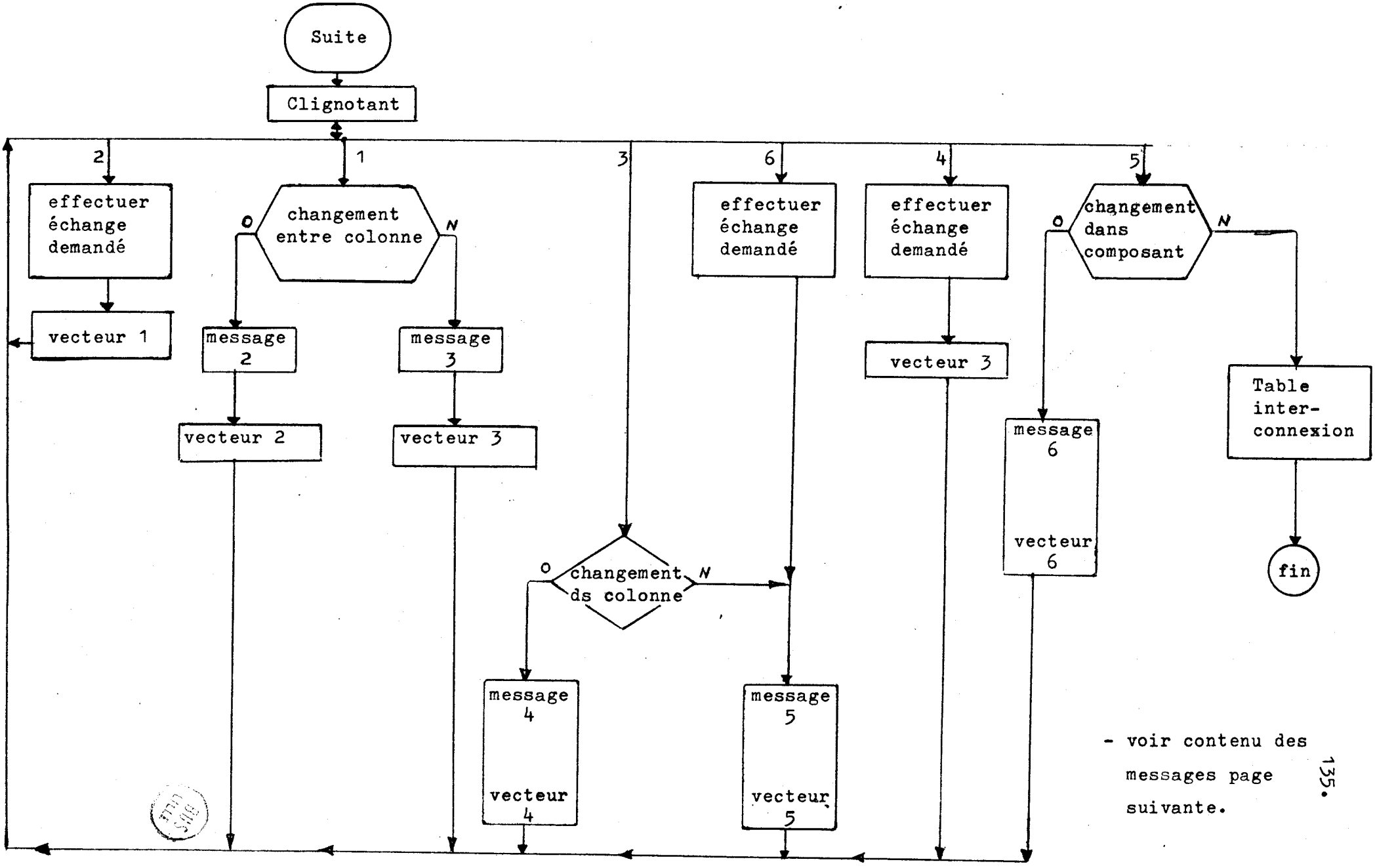
c) chaque composant sélectionné est représenté par son code

ASC II défini dans la table TCO.

Note: chaque fois qu'un composant est sélectionné, on change la valeur H'11' le précédant dans la table TC en H'12 + j'. j étant le n° de la colonne en cours de traitement.

3) Ordinogramme





- voir contenu des messages page suivante.

- Contenu des messages:

- message 1 : échange-t-on des composants de colonnes différentes ? '-'

- message 2 : noms des composants à échanger:
  '-' et '-'

- message 3 : échange-t-on des composants dans une colonne ? '-'

- message 4 : n° de la colonne ? '-'
  noms des composants:
  '-' et '-'

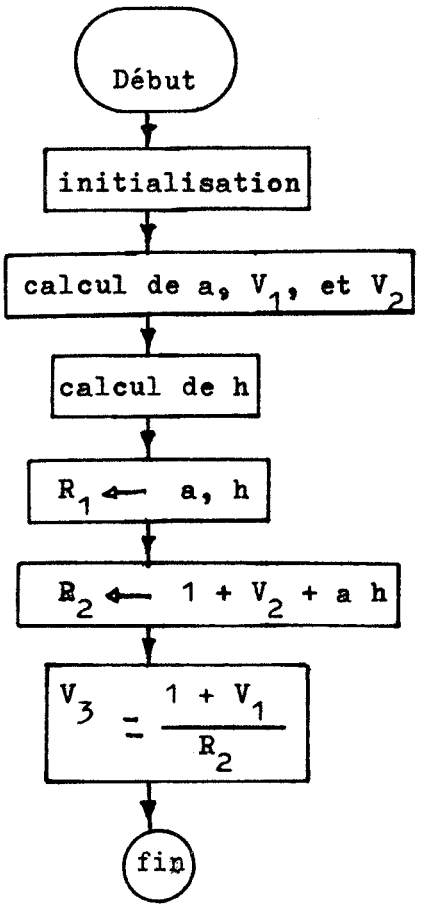
- message 5 : échange-t-on les fonctions dans un composant ?

- message 6 : nom du composant : '-'
  broches:
  '-' et '-'

Calcul de V_3

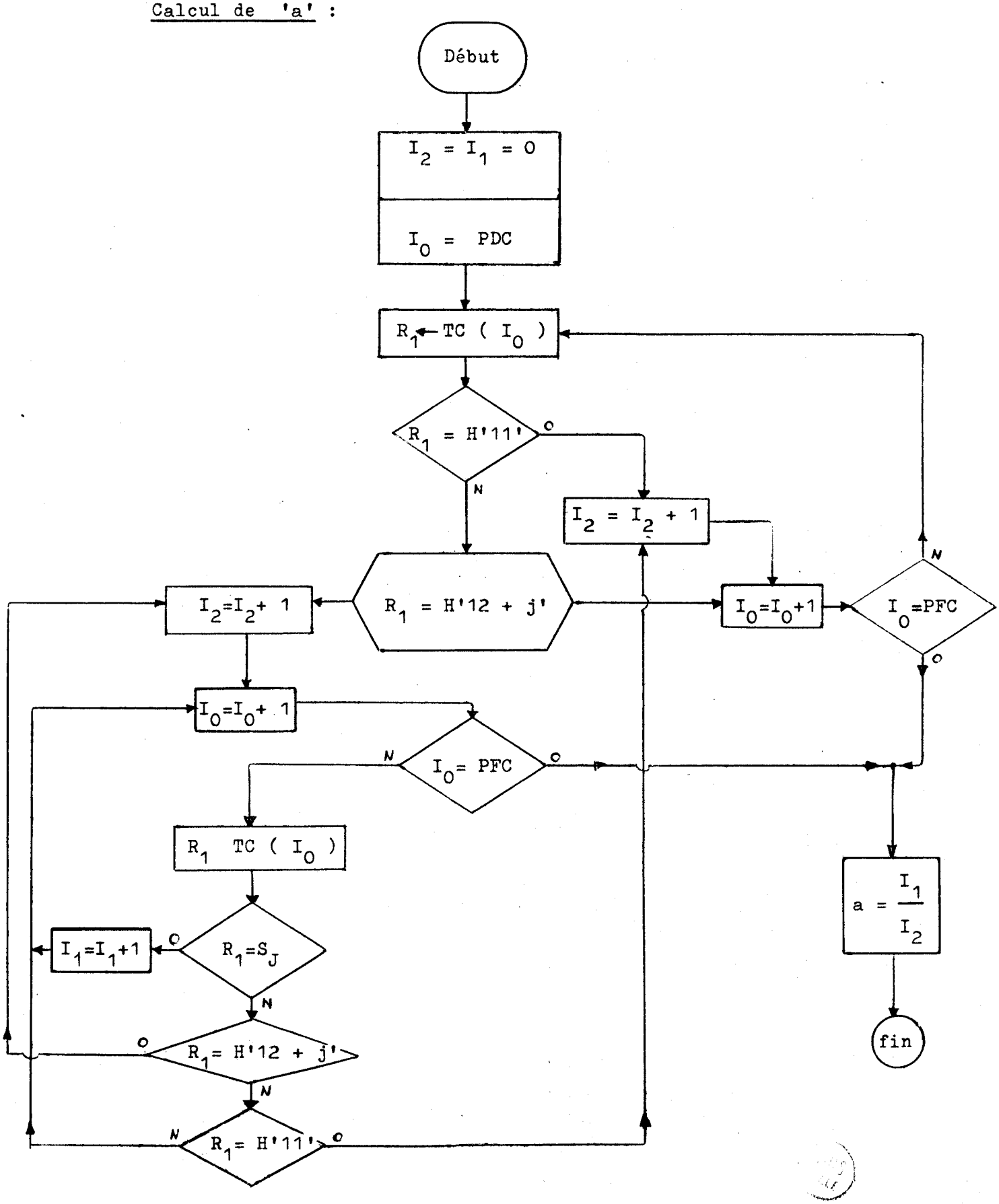
Rappelons que $V_3 = \dfrac{1 + V_1}{1 + V_2 + a\,h}$

(voir p 88)

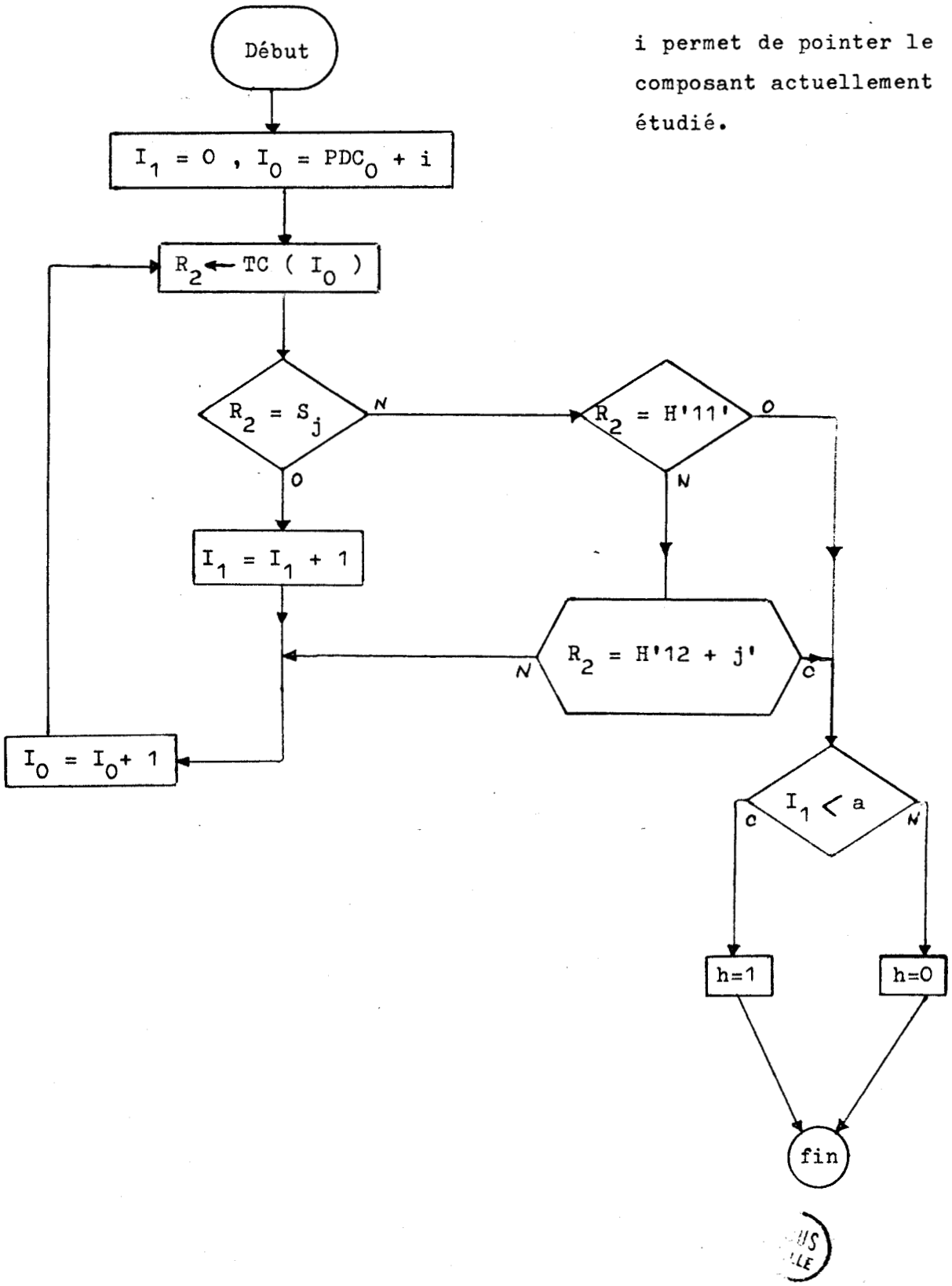


Nous donnons ci-après les méthodes de calcul de a, V_1 et V_2 ainsi que h . Notons que le calcul de a, V_1 et V_2 peut être simultané.

Calcul de 'a' :

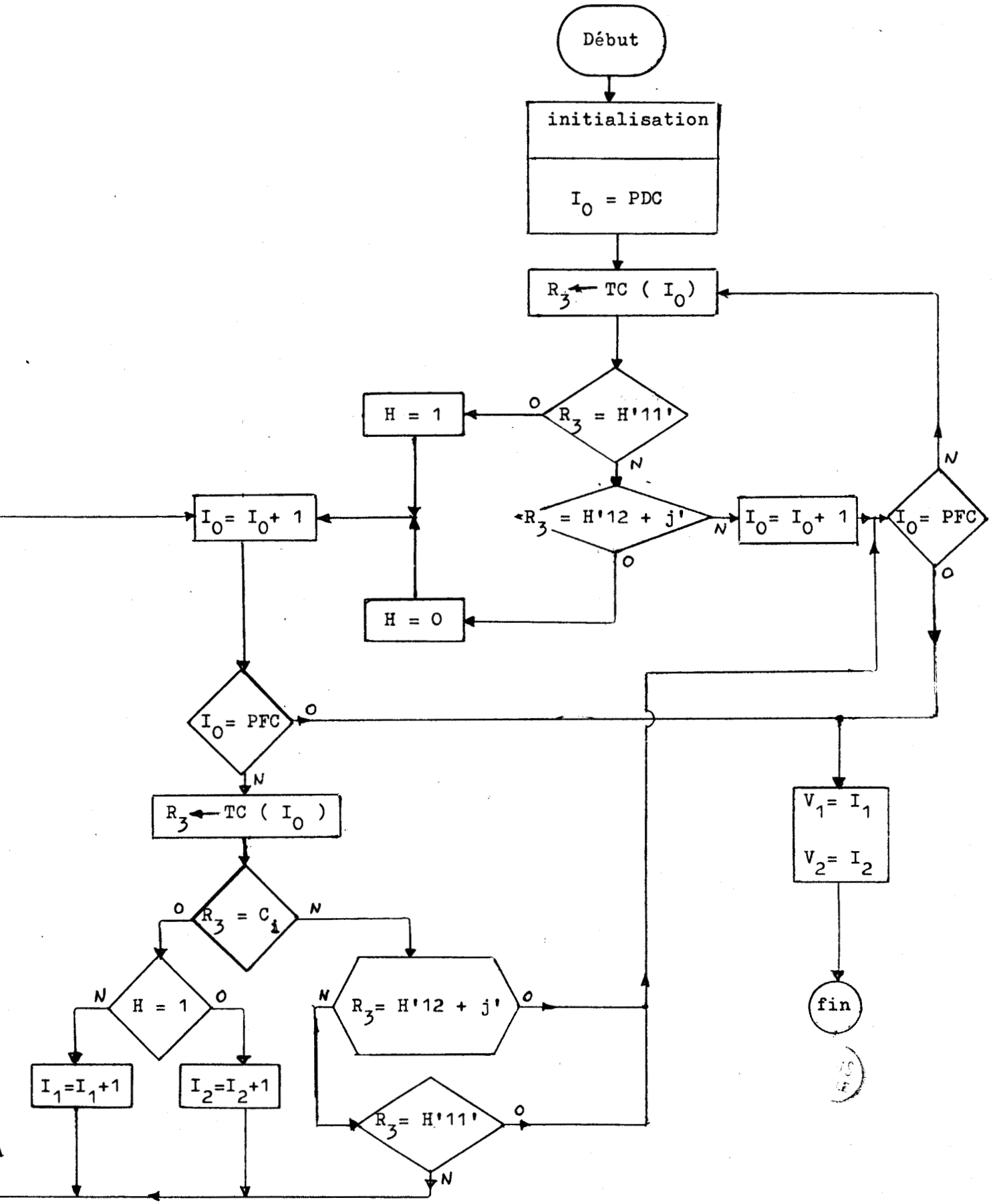

Début
$I_2 = I_1 = 0$
$I_0 =$ PDC
$R_1 \leftarrow TC\ (I_0)$
$R_1 =$ H'11'
O
N
$I_2 = I_2 + 1$
$R_1 =$ H'12 + j'
$I_2 = I_2 + 1$
$I_0 = I_0 + 1$
$I_0 =$ PFC
N
O
$I_0 = I_0 + 1$
$I_0 =$ PFC
N
O
R_1 TC (I_0)
$R_1 = S_J$
O
$I_1 = I_1 + 1$
N
$R_1 =$ H'12 + j'
O
N
$R_1 =$ H'11'
N
O
$a = \frac{I_1}{I_2}$
fin

Calcul de h :



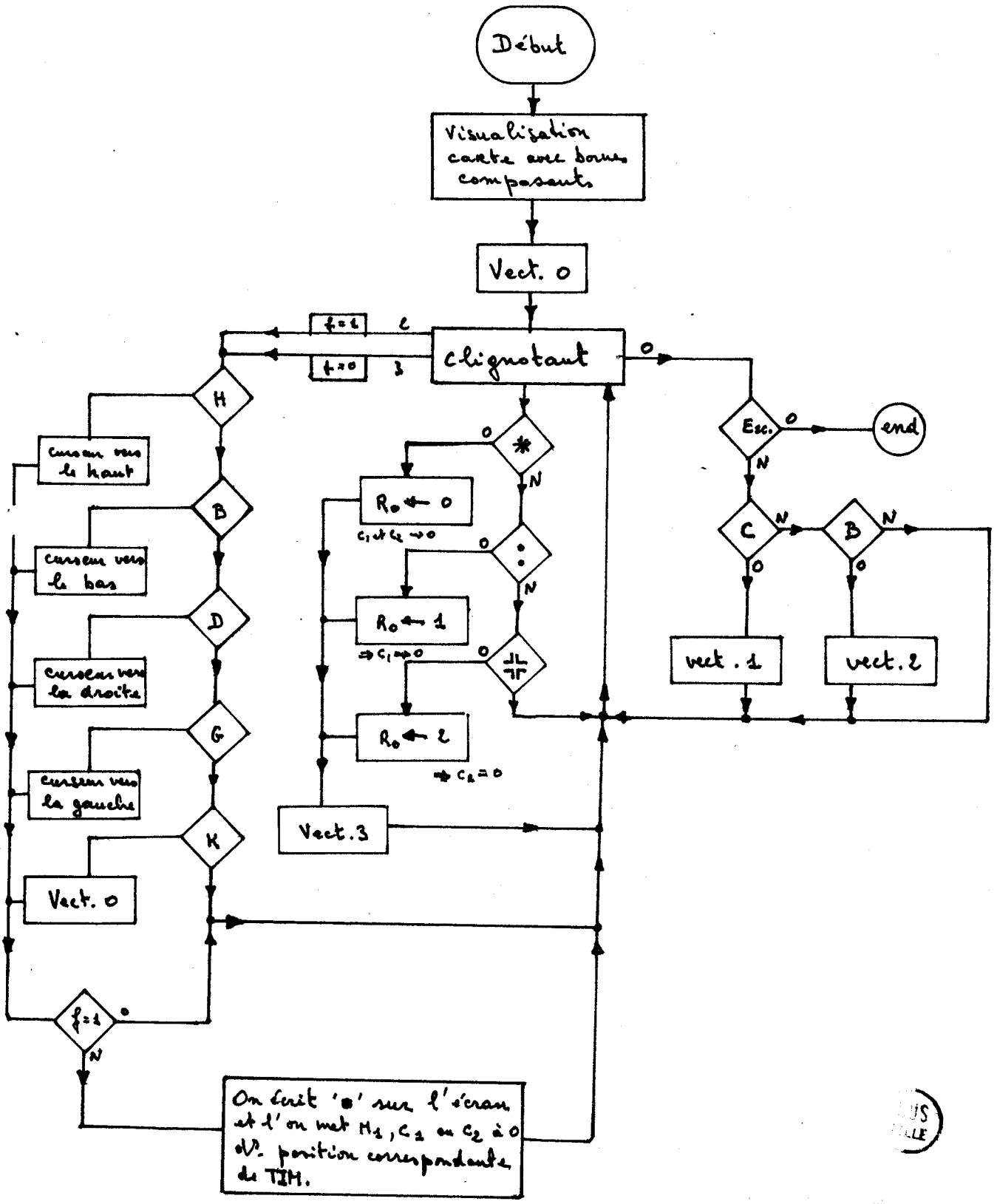
i permet de pointer le composant actuellement étudié.

Calcul de V_1 et V_2 :



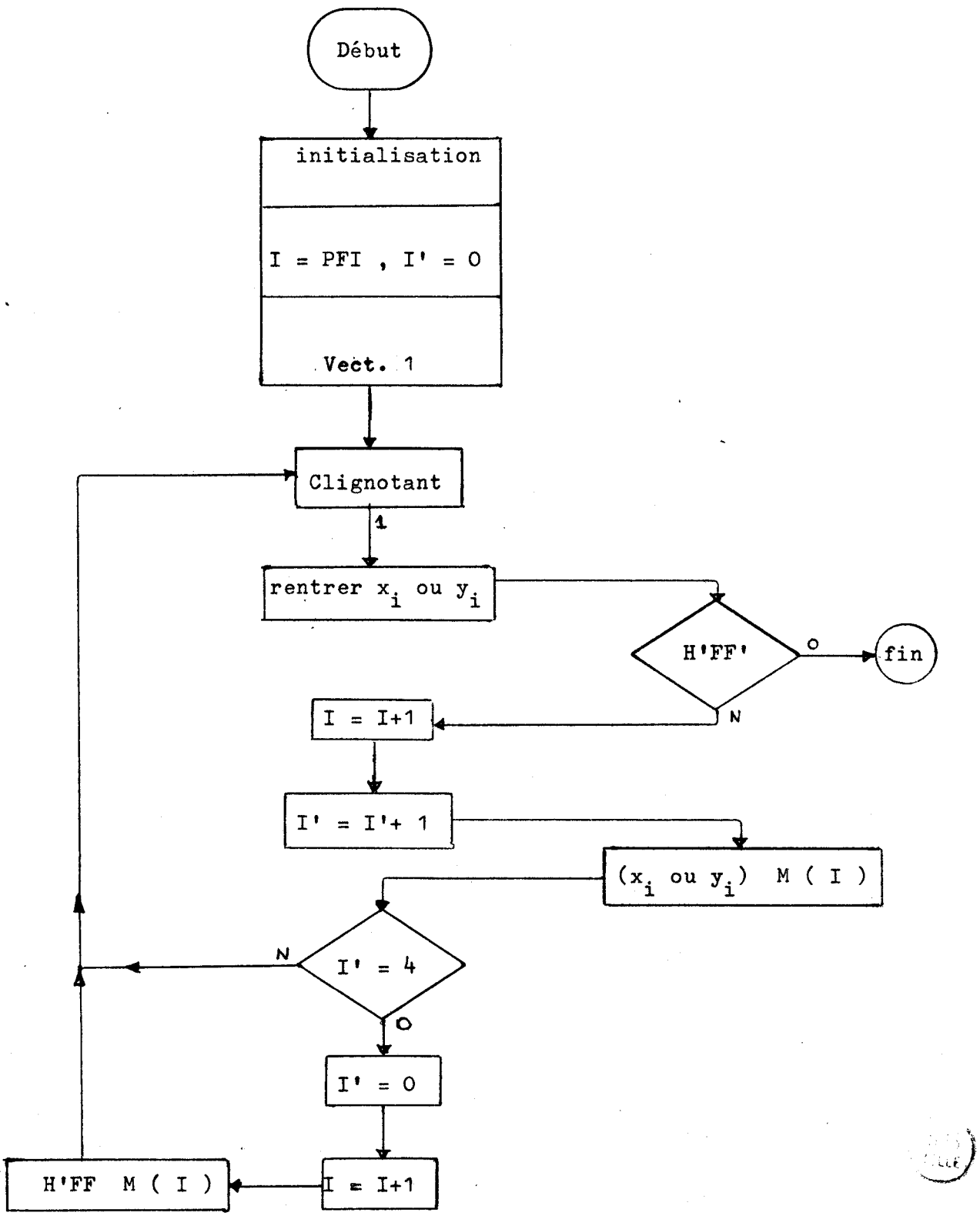
ANNEXE 4-2.

a) Ordinogramme de la procédure permettant de completer les "murs" sur la carte

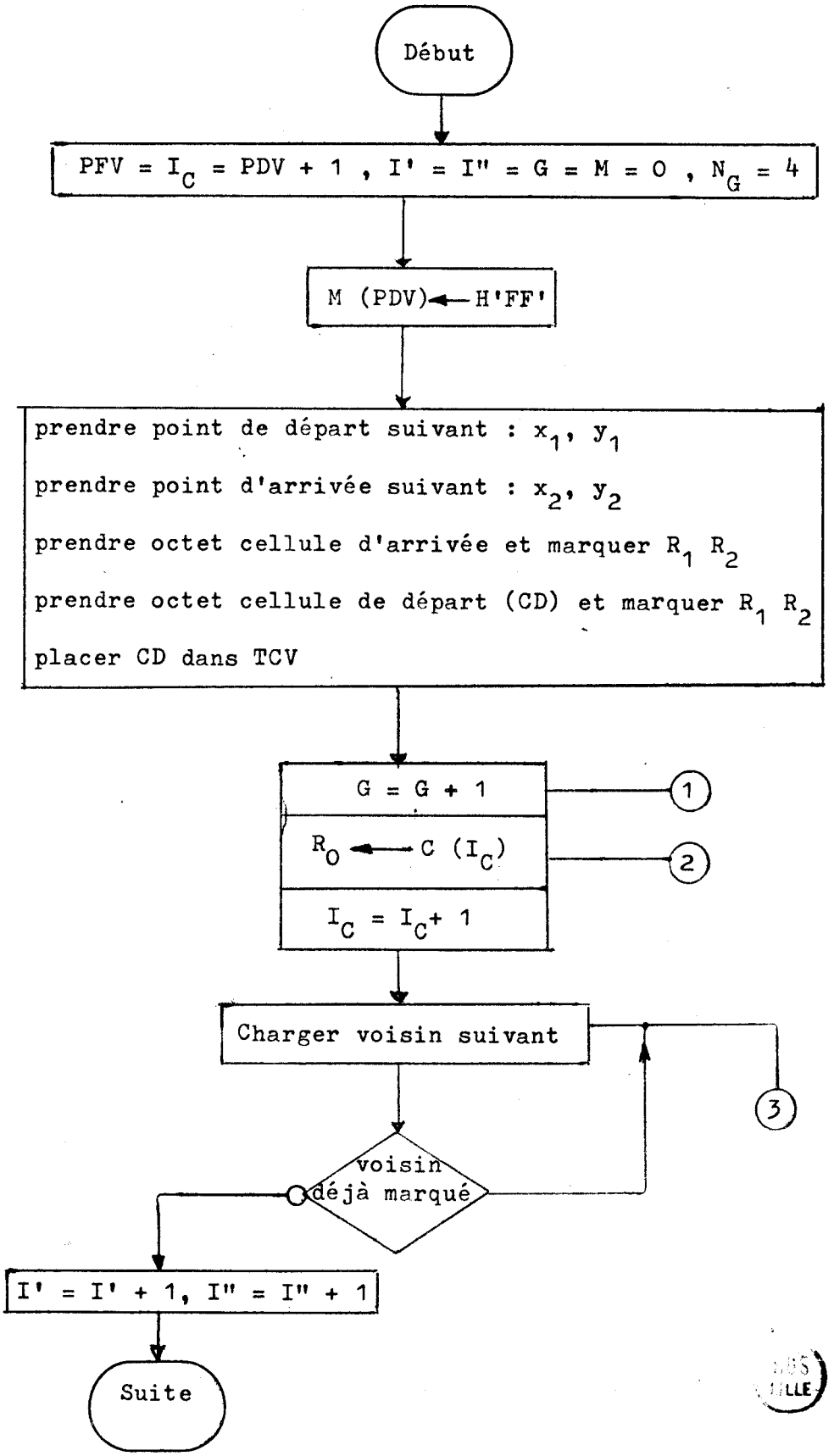


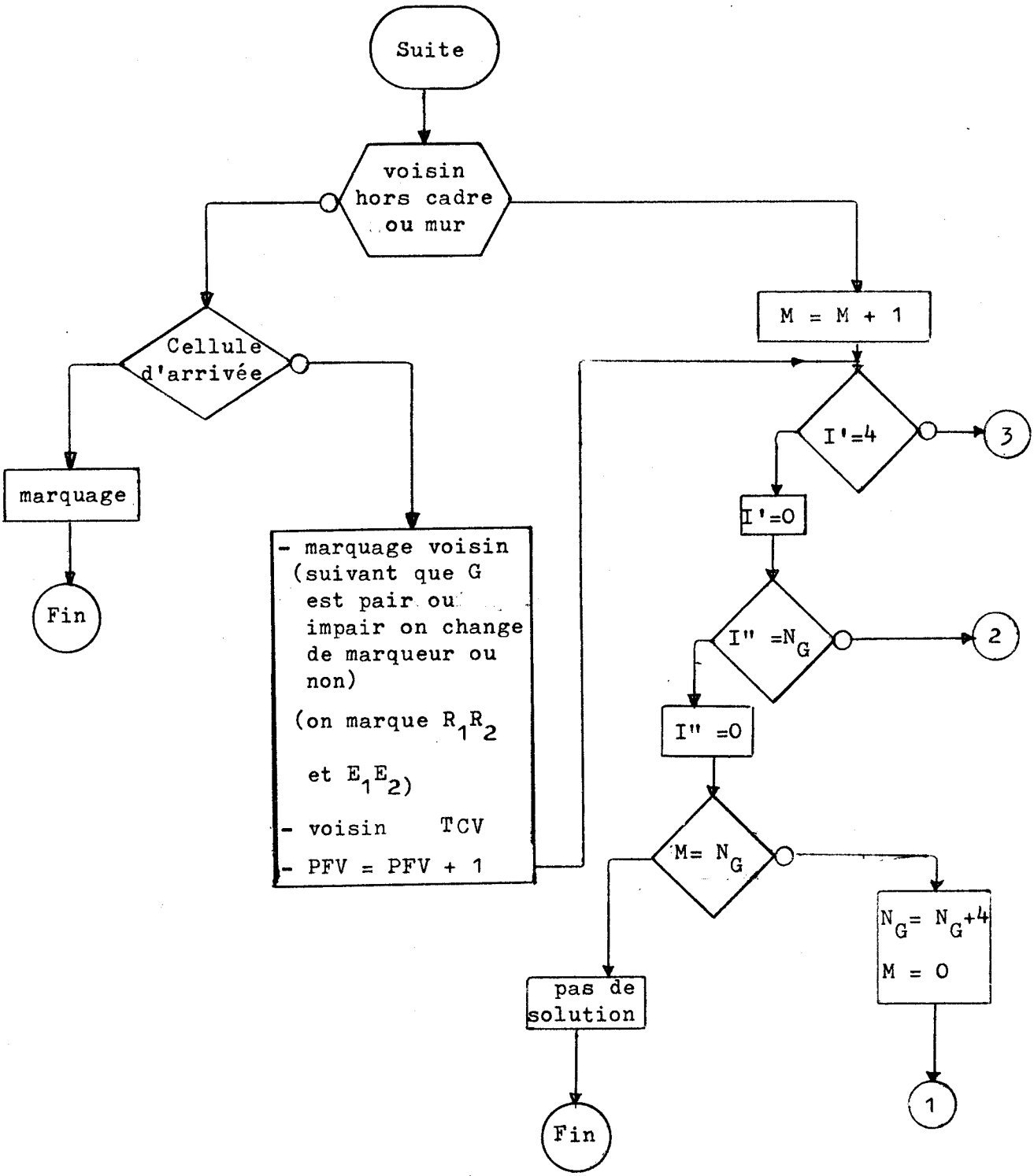
b) Ordinogramme permettant de rentrer de nouvelles interconnexions:

Lors de la phase 1 de l'ordinogramme précédent, on peut prévoir un 4ème caractère, le "blanc", qui reviendrait à "gommer" un "mur" sur l'écran et à réinitialiser l'octet correspondant de la table TMI.
On peut alors rentrer de nouvelles interconnexions.

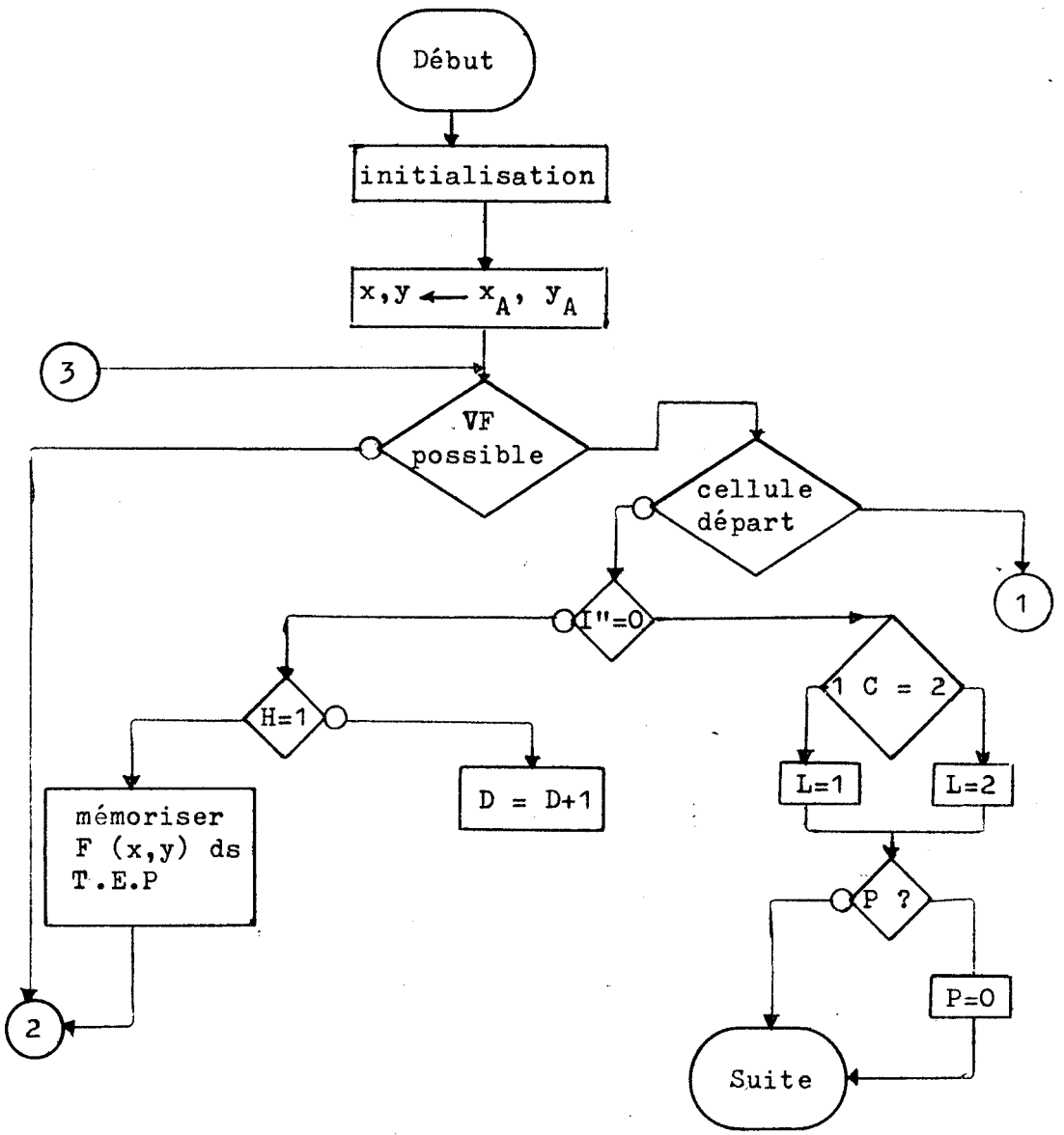


ANNEXE 4.3 Ordinogramme de recherche des voisins et marquage.




Suite
voisin hors cadre ou mur
Cellule d'arrivée
marquage
Fin
- marquage voisin (suivant que G est pair ou impair on change de marqueur ou non)
(on marque R_1R_2 et E_1E_2)
- voisin TCV
- PFV = PFV + 1
M = M + 1
I'=4
3
I'=0
I" =N_G
2
I" =0
M= N_G
pas de solution
Fin
N_G= N_G+4
M = 0
1

ANNEXE 4.4 Ordinogramme de recherche des chemins possibles et choix.



I" = nombre de voisins examinés par cellule

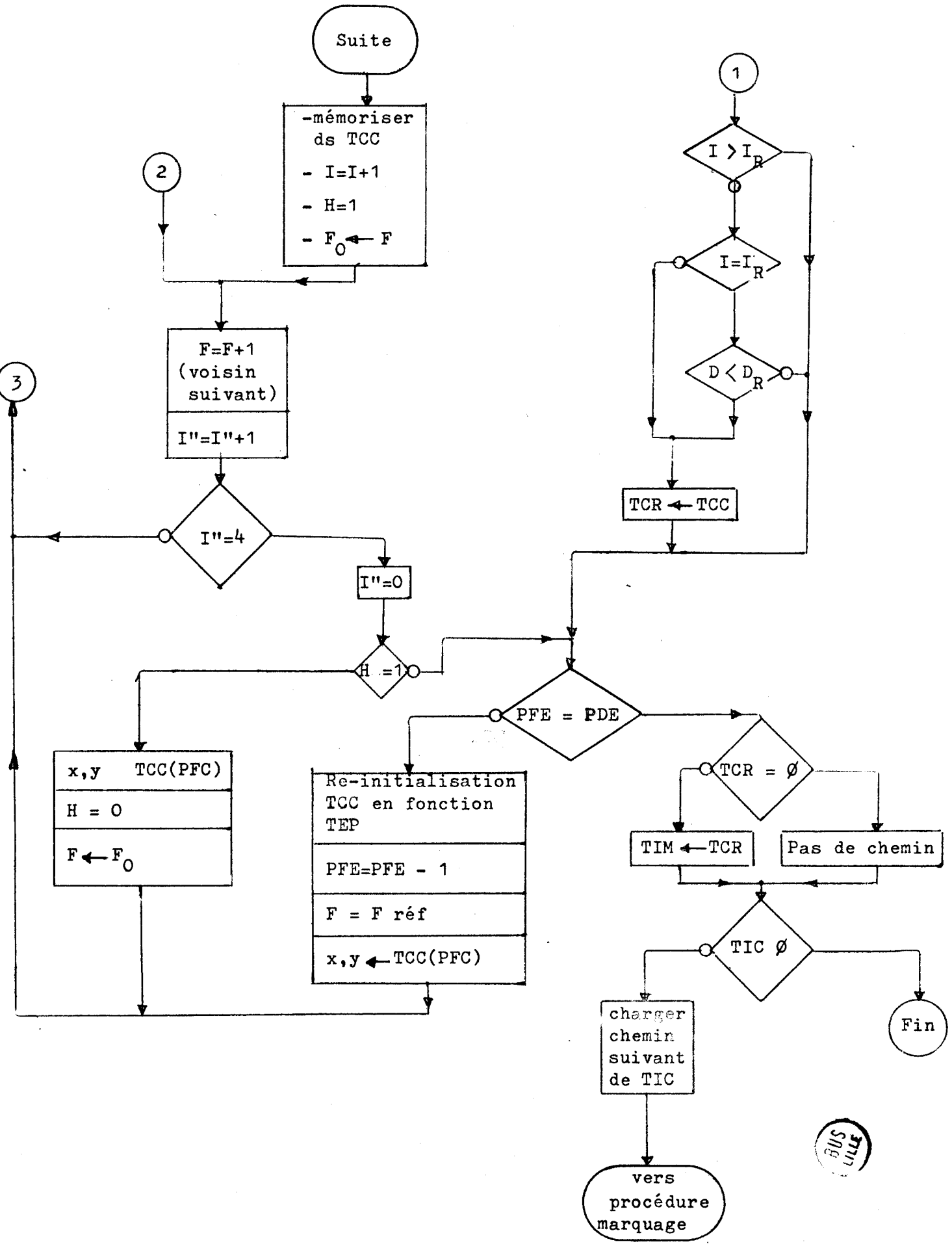
H = indicateur de ce qu'une cellule a été retenue ou non pour le chemin en construction

C = test concernant la couche sur laquelle continue le chemin

L = mémorise la couche

P = mémorise s'il y a changement de couche

T.P = table provisoire mémorisant les voisins de la cellule chargée en x, y susceptible de fournir des embranchements.


Suite
-mémoriser ds TCC
- I=I+1
- H=1
- F0 ← F
2
F=F+1 (voisin suivant)
I"=I"+1
3
I"=4
I"=0
H =1
x,y TCC(PFC)
H = 0
F ← F0
Re-initialisation TCC en fonction TEP
PFE=PFE - 1
F = F réf
x,y ← TCC(PFC)
1
I > IR
I=IR
D < DR
TCR ← TCC
PFE = PDE
TCR = Ø
TIM ← TCR
Pas de chemin
TIC Ø
charger chemin suivant de TIC
Fin
vers procédure marquage

BIBLIOGRAPHIE

1. Luther C. ABEL, "On the Ordering of Connections for Automatic Wire Routing"
IEEE Trans. on Computer, november 1972, pp 1227-1233

2. Sheldon B. AKERS, "Routing"
Chapter six, Volume one of "Design Automation of Digital Systems" edited by Melvin A. Breuer. 1972

3. Sheldon B. AKERS, "A modification of Lee's Path connections algorithm".
IEEE Tran. on Electronic Computers,
vol. EC-16, pp 97-98, February 1967

4. G. ALIA, G. FROSINI, and P. MAESTRINI, "Automated module placement and wire routing according to a structured biplanar scheme in printed boards".
Computer Aided Design, vol. 5, July 1973, pp 152-159

5. H. ANLAUFF, P. FUNK, and P. MEINEN, "PHPL - A New Computer Hardware Description Language for Modular Description of Logic and Timing".
Proceedings of the 4 th International Symposium on Computer Hardware Description Languages, Palo Alto, California, October 1979

6. H.M. BAYEGAN, and E. AAS, "An integrated system for interactive éditing of Schématic, Logic Simulation and PCB Layout Design ".
N° 15 Design Automation Conference (Proceedings) June 1978, Las Vegas. (IEEE)

7. Melvin A. BREUER, "Min-Cut Placement"
J. Design Automation Fault-Tolerant Computing
vol. 1 n° 4, pp 343-62, octobre 1977

8. Glenn R. CASE, and Jerry D. STAUFFER
A program to perform Logic Simulation and fault diagnosis

9. Robert C. CHEN, and James E. COFFMAN
MULTI-SIM, a dynamic multi-level simulator

10. Yaoha CHU, "CDL : Computer Design Language"
Computer December 1974

11. C. DEVALCK SCI-CARDS, note CD/CB - 3445
MBLE- Bruxelles, Avril 1980

12. D.L. DIETMEYER, "DDL : Digital System Design Language"
Computer December 1974

13. Donald L. DIETMEYER, "Logic Design of Digital Systems"
Allÿn and Bacon, 2e printing 1972

14. M. DUPONT et A. WOLFF, "Le Système Labyrus: description du programme de tracé TRACUS".
Revue Technique Thomson - CSF
vol. 6- n° 2, pp 425-455, juin 1974

15. M. DUPONT et A. WOLFF, "Le système Labyrus: description du programme d'implantation PAPYRUS".
Revue Technique Thomson - CSF
vol. 6- n° 2, pp 413-424

16. J.M. GEYER, "Connection routing algorithm for PCB's"
IEEE Trans. on circuit theory,
vol. CT-18, n° 1, pp 95-100, Jannary 1971

17. P.A. GRAWAL and M.A. BREUER, "Experiments with a Density Router for PC Cards".
IEEE Trans.on Computers,
vol. C-28 n°3, pp 262-267, March 1979

18. Maurice HANAN and Jerome M. KURTSBERG,
"Placement Techniques"
Chapter five, Volume one of "Design Automation of Digital Systems"
edited by Melvin A. Breuer, 1972

19. Reiner W. HARTENSTEIN and Ewald VON PUTTKAMER,
"KARL A Hardware Description Language as Part of a CAD Tool forLSI"
Proceedings of the 4 th International Symposium on Computer Hardware Description Language, IEEE, Palo Alto, California, October 1979.

20. G.R. HELLESTRAND, "MODAL A System for Digital Hardware Description and Simulation"
Proceedings of the 4 th International Symposium on Computer Hardware Description Languages, IEEE, Palo Alto, California, October 1979.

21. HIGHTOWER D W, "The interconnection Problem:A Tutorial".
Computer, 7,4, 1974, pp 18-32.

22. Dwight D. HILL, "ADILIB : A Modular, Strongly-Typed Computer Design Language".
Proceedings of the 4 th Internationnal Symposium on Computer Hardware Description Languages IEEE, Palo Alto, California, October 1979.

23. Frederick J. HILL, "A Design Language Bases Approach to Test Sequence Generation".
Computer June 1977

24. Frederick J. HILL, "AHPL : A Hardware Programming Language"
Computer December 1974.

25. Shingfat S C HUIGS and James H TRACEY, "An Interactive Computer Graphic Language for the Design and Simulation of Digital Systems".
Computer June 1977

26. Harry F. JORDAN and Burton J. SMITH, "The Assignment Statement in Hardware Description Languages".
Computer June 1977.

27. D. JOHNSON, "Simulation, verification and test package for logic design
Electronic Engineering January 1980.

28. A. KALETZKY and D.W. LEWIN,"Problem Orientated Language for Logic Design".
The Computer Journal 1975.

29. B.W. KERNIGHAN and S. LIN, "An efficient heuristic procedure for parttionning graphs".
Bell Sys. Tech. J.
vol. 49, Fecruary 1970, pp 291-308.

30. D.J. KINNIMENT and L.J. WESTON, "An evaluation of conventionnal and backtracking algorithms for routing printed circuit boards".
University of Manchester U.K.

31. Thaddeus KOBYLARZ, "Crosscompiler for hardware emulation".
Microprocessors and Microsystems vol. n° 9, November 1980

32. K.G. KULKARNI and A. PRABHAKAR, "Automated Routing of Printed Circuit Boards".
J. INSTN. Electronics Telecom. engrs. vol. 24, n° 3 et 4, 1978.

33. Douglas LEWIN, "Product Specification and Synthesis".
Department of Electrical Engineering and Electronics,
Brunel University Uxbridge, Middlesex.

34. G.J. LIPOVSKI, "Hardware description Languages: Voices from the Tower of Babel".
Computer June 1977.

35. R.W. MARCZINSKI and P. BAKOWSKI, "What Do the Computer Description Languages Describe ? ".
Proceedings of the 4 th International Symposium on Computer Hardware Description Languages IEEE, Palo Alto, California, October 1979.

36. P. MEINEN, "Formal Semantic Description of Register Tranfer Language Elements and Mechanized Simulator Construction".
Proceedings of the 4 th International Symposium on Computer Hardware Description Languages IEEE, Palo Alto, California, October 1979.

37. K. MIKAMI and K. TABUCHI, "A computer program for optimal routing of printed circuit conducters".
IFIPS
Proc. 1968, pp 1475-1478.

39. Leah MORY-RAUCH, "Pin Assignment on a printed circuit Board".
n° 15 Design Automation Conference Proceedings, June 1978,
Las Vegas (IEEE).

40. Dan NASH, "Topic in Design Automation Data Bases"
n° 15 Design Automation Conference Proceedings, June 1978
Las Vegas (IEEE).

41. G.R. PETESON, "Teaching Digital Systems Design Whith a Hardware Description Language".
Proceeding of the Eighth Annual Frontiers in Education Conference
Lake Buena Vista, Fl. USA., Oct. 1978. (IEEE).

42. POUILLONG, Note BC-DT, GP/CB - 3303
MBLE Bruxelles, Avril 1980.

43. L.S. PUGH, "An improvement in printed circuit board routability using a maze-running algorithm".
Electronics letters 5 th January 1978, vol. 14 n° 1, pp 8.9

44. Benson H. SCHEFF, Stephen P. YOUNG, "Design Automation of digital systems".
Chapter three
edited by Melvin A. Breuer.

45. Donald M. SCHULER, "A Language for Modeling the Functional and Timing Characteristics of Complex Digital Components for Logic Simulation".
Proceedings of the 4 th International Syposium on Computer Hardware Description Languages IEEE, Palo Alto, California, October 1979.

46. Sajjan G. SHIVA, "Use for DDL in an Automatic LSI Design and Test System".
Proceedings of the 4 th International Symposium on Computer Hardware Description Languages IEEE, Palo Alto, California, October 1979.

47. Dan SIEWIOREK, "ISP : Instruction Set Processor".
Computer December 1974.

48. Dan SIEWIOREK, "PMS : Processors, Memories and Switches".
Computer December 1974

49. Stephen Y.H SU, "A Survey of Computer Hardware Description Languages in the USA".
Computer December 1974

50. J.H. STEWART, "LOGAL : A CHDL for Logic Design and Synthesis of Computers".
Computer June 1977.

51. M. TOKORO, M. SATO, M. ISHIGAMI, E. TAMURA, T. ISHIMITSU and H. OHARA.
A module Level Simulation Technique for systems composed of LSI's and MSI's.

52. John WALLACE, Alice C. PARKER, "SLIDE : An I/O Hardware Descriptive Language".
Proceedings of the 4 th International Symposium on Computer Hardware Description Languages IEEE, Palo Alto, California, October 1979.

53. Conception assistée par ordinateur pour circuits imprimés.
Bulletin RTC n° 47, Decembre 1979.

54. C.Y. LEE, "An algorithm for path connections and its applications
IEEE Trans. on Electronic Computer
vol. EC-10, pp 346-365, September 1961

55. Sajjan G. SHIVA, "Computer Hardware Description Languages - a Tutorial"
Proceedings of the IEEE. December 1979.