Formalisme DELTA : un outil de description logique pour la synthèse automatique dans la conception des machines séquentielles synchrones

Mohamed Nemmour

▶ **To cite this version:**

Mohamed Nemmour. Formalisme DELTA : un outil de description logique pour la synthèse automatique dans la conception des machines séquentielles synchrones. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1981. Français. NNT : . tel-00297303

HAL Id: tel-00297303
https://theses.hal.science/tel-00297303

Submitted on 15 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE TROISIEME CYCLE

par

Mohamed NEMMOUR

FORMALISME DELTA : UN OUTIL DE DESCRIPTION LOGIQUE POUR LA SYNTHESE AUTOMATIQUE DANS LA CONCEPTION DES MACHINES SEQUENTIELLES SYNCHRONES.

Thèse soutenue le 3 décembre 1981 devant la Commission d'Examen :

Monsieur	F. ANCEAU	: Président
Messieurs	M. DEPEYROT G. MICHEL G. NOGUEZ J.M. RATA	Examinateurs

Je tiens à remercier,

Monsieur F.ANCEAU, Professeur à l'Institut National Polytechnique de Grenoble, qui a bien voulu me faire l'honneur de présider le jury de cette thèse, pour ses conseils et ses encouragements qui ne faillirent jamais.

Messieurs M.DEPEYROT, Directeur des Projets à EFCIS, G.MICHEL, Chef du Departement d'Architecture des Systèmes, CNET/Meylan, G.NOGUEZ, Professeur à l'Institut de Programmation, Paris VI, et JM.RATA, Conseiller Technique du Departement TIEM, EDF/Clamart, qui ont bien voulu accepter de participer au jury de cette thèse, malgré leurs nombreuses occupations.

Tous les Membres de l'Equipe de Recherche en Architecture des Ordinateurs, dont les nombreuses discussions et la complicité ont facilité la réalisation de cette étude.

Toutes les Personnes qui ont contribué à la dactylographie de ce document, notamment Madame H.DIAZ pour sa gentillesse et sa bonne humeur éternelle.

Le Service de Reprographie de l'IMAG dont la compétence et l'efficacité n'est plus à présenter.

TABLE DES MATIERES

V. QUELQUES REFLEXIONS SUR L'ASPECT CAO

1. STRUCTURES D'IMPLANTATION SUR ORDINATEUR DES DESCRIPTION DELTA
2. NOTION DE BIBLIOTHEQUE D'OUTIL DE CAO
3. FORMALISME DELTA ET STRATEGIES DU TRAITEMENT AUTOMATIQUE

Partie.3: CARACTERISTIQUES GENERALES DES CIRCUITS SEQUENTIELS

Presentation generale

VI. CARACTERISTIQUES GENERALES DES MACHINES SEQUENTIELLES

1. CONCEPTS DE BASE

11.MACHINE SEQUENTIELLE

12.CIRCUIT COMBINATOIRE

13.NOTION DE PARTIE OPERATIVE ET DE PARTIE CONTROLE

2. PARTIE CONTROLE

21.MECANISME GENERAL DE CONTROLE

211.NOTION DE SEQUENCEUR ET D'INTERFACE DE COMMANDES

212.TECHNIQUE DE CODAGE DES ETAS INTERNES

22.ASPECT MATERIEL DES PARTIES CONTROLE

221.STRUCTURES CABLEES

a) CARACTERISTIQUES PARTICULIERES DES STRUCTURES CABLEES

b) NOTION DE LOGIQUE POSITIVE ET DE LOGIQUE NEGATIVE

222.STRUCTURES A ALGORITHME ENREGISTRE

a) CARACTERISTIQUES PARTICULIERES DES STRUCTURES MICROPROGRAMMEES

b) CARACTERISTIQUES PARTICULIERES DES STRUCTURES A BASE DE PLA

3. PARTIE OPERATIVE

31.NOTION D'UNITE DE TRAITEMENT

32.MECANISME GENERAL DU TRAITEMENT DES DONNEES

Partie.4: PROJET 6800.S

Presentation generale

VII. GENERALITES

1. CARACTERISTIQUES TECHNIQUES
2. JEU D'INSTRUCTIONS

VIII. ARCHITECTURE INTERNE

1. PARTIE OPERATIVE
11. ARCHITECTURE DE BASE
12. MECANISME D'EXECUTION PIPE-LINE

2. PARTIE CONTROLE
21. ARCHITECTURE DE BASE
22. FONCTIONNEMENT DU SEQUENCEUR
23. FONCTIONNEMENT DE L'INTERFACE DE COMMANDES

IX. POINTS CRITIQUES DE FONCTIONNEMENT

1. SEQUENCE INTERNE DE DEMARRAGE A LA MISE SOUS TENSION
11. CONDITION INITIALE.1
12. CONDITION INITIALE.2

2. SEQUENCE INTERNE DE DEMARRAGE LORS D'UNE INITIALISATION
21. INITIALISATION A LA MISE SOUS TENSION
22. INITIALISATION PENDANT LE FONCTIONNEMENT

3. TRAITEMENT AU NIVEAU DU SYSTEME D'INTERRUPTIONS
31. MECANISME DE VALIDATION DES SIGNAUX D'INTERRUPTION
32. BARRIERE D'ECHANTILLONNAGE DU SIGNAL RESET
33. BARRIERE D'ECHANTILLONNAGE DU SIGNAL HALT

XII. CONCLUSION

XIII. ANNEXES

Annexe.1: DESCRIPTION DELTA DE LA PARTIE CONTROLE 6800/6800.S

1. DESCRIPTION DELTA DU SEQUENCEUR
2. DESCRIPTION DELTA DE L'INTERFACE DE COMMANDES

Annexe.2: CODES INVALIDES DES MICROPROCESSEURS 6800

1. SEQUENCEMENT GENERAL
11. SEQUENCE ACQUISITION
12. SEQUENCE ADRESSAGE

2. TRAITEMENT DES CODES INVALIDES
21. CARACTERISTIQUES GENERALES
22. CODES INVALIDES DU TYPE.1
23. CODES INVALIDES DU TYPE.2
24. CODES INVALIDES DU TYPE.3

3. MONTAGE EXTERNE POUR SUPPRIMER LES ANOMALIES 1, 2 DU 6800

4. NOTATIONS ET SYMBOLES UTILISES DANS LE PROJET 6800.S

Annexe.3: EVALUATIONS CONCRETES SUR LE 6800.S

1. BROCHAGE DU 6800.S
2. EVALUATION LOGIQUE SUR LE BLOC (ANOMALIE)
3. EVALUATION ELECTRIQUE
31. BLOC (ANOMALIE)
32. BASCULE NMI

4. EVALUATION TOPOLOGIQUE
41. CORRECTION DE L'ANOMALIE.2
42. TOPOLOGIE DU 6800.S

INTRODUCTION

I. INTRODUCTION

Une étape importante a été franchie dans la maitrise des problèmes technologiques. Ceci laisse prévoir un formidable développement des circuits intégrés (déjà amorcés avec le LSI).

Les prévisions les plus moderées (Fig. 1) laissent à penser que le nombre de transistors par circuit intégré serait de l'ordre du million (10^6 éléments actifs) dans les dix prochaines années.

Ces chiffres sont loins de constituer une limite pour les circuits des années 2000.

Toutes ces possibilités apportées par les progrès technologiques permettent une nette amélioration quant aux performances des circuits intégrés (C.I) et encouragent (fortement) à élargir leur champ d'application.

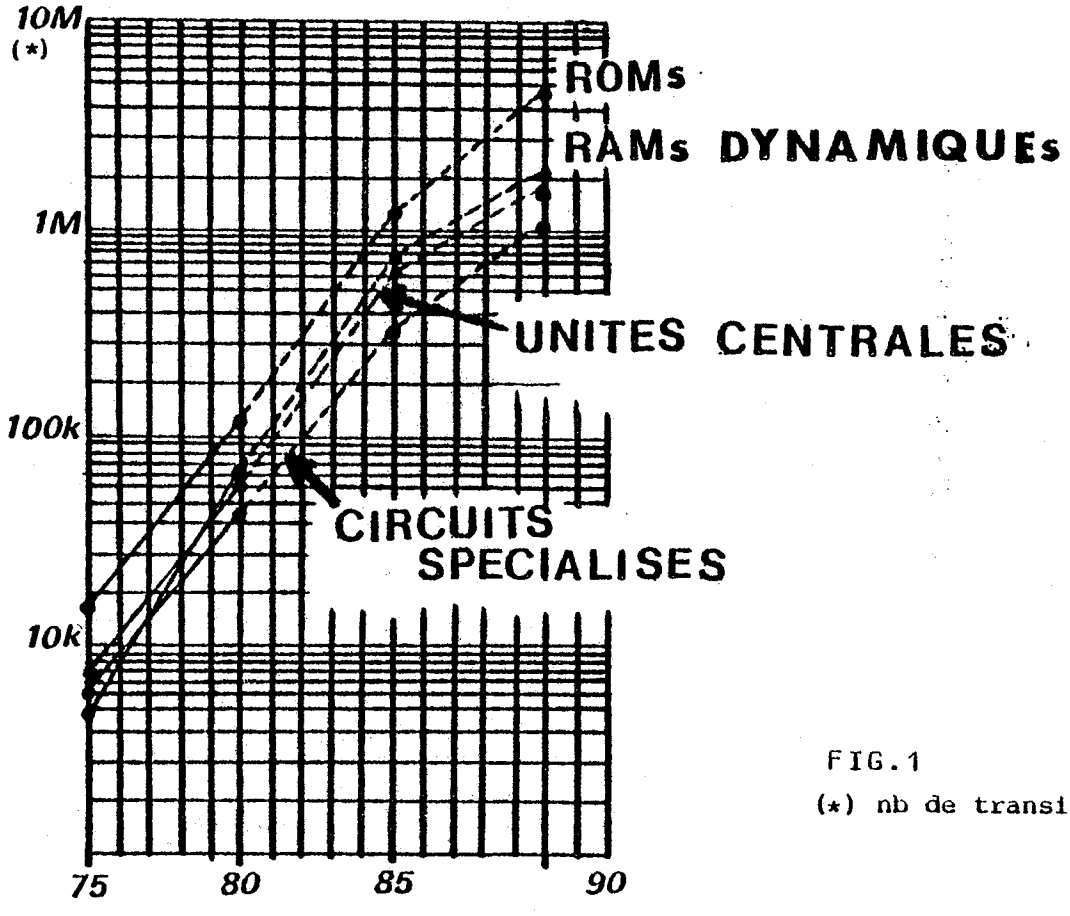


FIG.1
(*) nb de transistor/CI

D'autres chiffres [22] montrent que le coût de la conception avec les techniques actuelles (en grande partie manuelles) croit de plus en plus. Ainsi un circuit intégré de 100.000 transistors nécessite 40 Hommes/Années pour sa conception et autant pour sa mise au point finale (validation, test...)

Vu sous un autre angle, ceci signifie qu'il faudrait quarante années de la vie d'un concepteur expérimenté, qui serait emmené à travailler seul sur un tel projet, pour concevoir un tel circuit (10^5 transistors) et autant d'année pour sa mise au point finale (à supposer qu'il vive centenaire).

Ces chiffres parlent d'eux mêmes, alors que la concurrence industrielle veut qu'il faut toujours aller plus loin dans les performances, plus rapidement dans la réalisation et à des coûts les plus bas.

Il est clair que les techniques actuelles de la conception des C.I. ne permettent pas de satisfaire de telles exigences et que des modifications radicales, quant aux moyens à utiliser pour réaliser les futurs C.I. sont indispensables.

L'automatisation de la conception des C.I. constitue une solution efficace pour répondre à une grande partie de ces problèmes économiques.

Cette évolution dans la conception des C.I. détermine un lieu (plus étroit encore) entre le concepteur et la machine. Ceci se caractérise par la diminution, voir la suppression de l'intervention humaine (directe) dans les différentes étapes de la conception.

La réalisation par la machine des différents traitements, sous le contrôle du concepteur, facilite énormément la conception et diminue par la même occasion les risques d'erreurs et donc le coût total des projets.

Pour ce qui est de l'automatisation de la conception, deux approches sont possibles :

- ou bien avoir des outils de CAØ indépendants, très performants, et bien spécialisés qui s'appliquent à des projets bien spéciaux,

- ou bien avoir un ensemble d'outils de CAO (Tous aussi performants) qui opèrent sur des données communes et s'appliquent à tout un ensemble de projets differents .

La deuxième solution, à laquelle nous adhérons, semble être la plus intéressante et la plus efficace (44) (45) :

- comptabilité des différents outils de CAØ,
- structures de données connues,
- réduction importante du volume total du logiciel (*)

Dans la relation Homme/machine, nous pensons qu'il faut pouvoir adapter les outils de CAO aux besoins du concepteur des C.I. plutôt que l'inverse.

Cette automatisation poussée, nécessite des moyens de description des circuits séquentiels adéquats pour laisser au concepteur toutes les facilités d'expressions.

De plus, ces outils de descriptions doivent répondre au désir profond des concepteurs qui est :

La transposition des techniques (de toutes les techniques) de la conception manuelle au niveau des outils de CAO d'une manière simple et directe.

---

* Possibilité d'utilisation d'une structure modulaire dans les programmes.

Notre démarche à travers cette étude s'inscrit dans ces grandes lignes.

La présente thèse, qui comporte quatre parties principales, aborde les différents aspects liés à la synthèse automatique dans la conception des circuits intégrés :

- définition d'un formalisme de description (partie 1) ;
- organisation du système d'outils de CAO (partie 2) ;
- caractéristiques des mécanismes généraux des circuits séquentiels (partie 3) ;
- problèmes liés aux techniques manuelles de la conception des C.I. (partie 4).

Un résumé est présenté au début de chacune de ces quatres parties.

PARTIE.1

FORMALISME DELTA

PRESENTATION GENERALE

Nous analysons les motivations générales de l'étude en mettant l'accent sur les caractéristiques qui ont déterminé le choix des primitives du formalisme DELTA.*

L'idée de correspondance directe et sans ambiguité entre une description algorithmique et sa représentation matérielle (informatisable via DELTA) est introduite, en se basant sur le fait qu'il faut pouvoir adapter les outils de CAO aux besoins du concepteur des circuits intégrés plutôt que l'inverse.

Nous définissons ensuite le formalisme DELTA et, à travers des exemples concrets, nous montrerons la grande souplesse offerte par DELTA aussi bien pour la description des circuits que pour la réalisation des outils de CAO.

---

* initiales de: Description Logique pour le Traitement Automatique

II - MOTIVATION

Il est bien connu que le coût de la conception manuelle dans l'industrialisation d'un circuit intégré est important.

Le besoin en outils de CAO pour les concepteurs de circuits intégrés croît de plus en plus. Plusieurs outils de description ont été définis pour permettre la synthèse automatique dans la conception des machines séquentielles.

La plupart de ces outils de description se basent sur la notion de "boîte noire"*: DDL, LDL , CASSANDRE, CDL ...

Cette notion de boîte noire, malgré les avantages réels qu'elle apporte quant à la clarté et à la simplification des descriptions matérielles, renferme tout un ensemble de sous-entendu et ne permet pas toujours à l'utilisateur d'exprimer toutes les caractéristiques matérielles de l'objet qu'il décrit. Aussi, à un certain stade de la conception, le concepteur se trouve handicapé par le fait qu'il n'a pas de moyens souples et directs d'accéder à la structure fine des objets (même s'il la devine) car tous les détails de l'objet décrit lui sont masqués, les boîtes qu'il manipule étant opaques.

En plus de ces inconvénients, les outils de description du type RTL*** ne sont pas adaptés à la description des caractéristiques matérielles des fonctions de contrôle dans un circuit séquentiel.

Un certain nombre d'outils spécialisés dans la description de fonctions de contrôle ont été définis, tels que : RDP, GDB, LORES, GRAFCET...

Malheureusement, ceci n'a pas permis de résoudre tous les problèmes. En effet, en plus de leur caractère spécifique, l'inconvénient majeur de ces outils de description est qu'ils sont trop lourds à manipuler et ne permettent pas (également) d'exprimer les différentes caractéristiques matérielles d'une

---

* Représentation symbolique d'un objet matériel (REGISTRE, ADDITIONNEUR)

*** de l'anglais Register Transfer Level (niveau de transfert registre)

manière simple et souple.

Ceci est généralement compensé par un développement important "d'artifices" au niveau du logiciel pour résoudre tel ou tel problème de description matérielle.

Ainsi, les modèles d'outils de CAO définis sur des structures trop abstraites, ne correspondent pas toujours à la réalité matérielle et par là même, ne répondent pas au besoin profond du concepteur de circuits intégrés qui est : la transposition des Techniques, DE LA CONCEPTION MANUELLE, au niveau des outils de CAO d'une manière simple et directe.

Nous nous garderons d'ailleurs bien, dans ce qui suit, de vouloir trancher les querelles d'écoles relatives aux solutions (les meilleures) apportées par tel ou tel aspect du logiciel ou du matériel.

Nous pensons quant à nous, qu'il n'est pas indispensable de s'éloigner de la structure matérielle pour formaliser tel ou tel problème matériel à l'aide de l'outil informatique. Une solution simple et efficace pour pallier cette difficulté consiste à définir un formalisme qui permet le traitement informatique (donc le traitement automatique par la machine) tout en conservant les traits fondamentaux de l'aspect architectural des objets.

Le formalisme DELTA a été conçu dans cet esprit. Ainsi, la conception et la realisation des outils de CAO se trouve largement facilitée par le fait que les descriptions DELTA sont etroitement liées aux caractéristiques materielles des objets decrits .

III - CONCEPTS DE BASE

1 - Notion d'ETA

Un ETA* est défini par un 3- uplet

$$< E, T, A >$$

dans lequel :

E est l'ensemble des entrées de l'ETA

T est l'ensemble des fonctions booléennes de "transfert" (affectation, connexion) qui permet le positionnement de la sortie de l'ETA.
L'ensemble T est appelé : barrière d'activation.

A est l'emplacement de l'unique sortie de l'ETA appelée : PLACE
La nature et la structure matérielle de la place d'un ETA varie selon l'objet qui est décrit (élément de mémorisation, élément de connexion).

La place porte une étiquette qui sert d'identificateur à l'ETA et la valeur logique qui lui est associé, traduit l'état d'activation de celui-ci :

un ETA M est dit actif si (M=1)
un ETA M est dit inactif si (M=0)

Les différentes caractéristiques d'un ETA sont destinées à la description d'un objet matériel (dans un circuit séquentiel synchrone) aussi bien du point de vue de sa structure que de celle de sa fonction.

---

* Le symbole ETA est issu de : Entrées Transfert Activation ; cela peut être aussi le diminutif de (ETAt, ETApe, ETAge (registre,...)). Pour indiquer qu'il sert aussi bien à la description des fonctions de contrôle qu'à celle des opérateurs dans un circuit séquentiel synchrone.

2 - Primitives de base

Il existe deux familles de primitives dans le formalisme DELTA :

- les ETA séquentiels qui permettent la description des circuits séquentiels* (tels que : Bascule RS, Registre,...)

- les ETA combinatoires qui permettent la description des circuits combinatoires*

Aucune indication purement technologique ou électrique n'est introduite au niveau des primitives de base.

Notons que, pour une question d'habitude d'usage, la plupart des symboles/appelations utilisés dans le formalisme DELTA sont ceux utilisés dans les langages RTL. Il en est de même pour les différentes formes languistiques qui sont utiliséespour représenter des primitives de base du formalisme :

. instruction d'affectation***

. instruction de connexion***

Cependant, les concepts de base restent propres au Formalisme DELTA.

Les principaux symboles sont :

+	Opérateur logique OU	< =	symbole d'affectation
.	Opérateur logique ET	: =	symbole de connexion
¬	Opérateur logique NON	<...>	expression conditionnelle si (...) alors

---

* voir définition (PARTIE.3)

***voir paragraphes suivants

2.1. - ETA séquentiel synchrone

De manière concise, l'ETA séquentiel synchrone est appelé ETA synchrone.
Sa représentation graphique est la suivante :

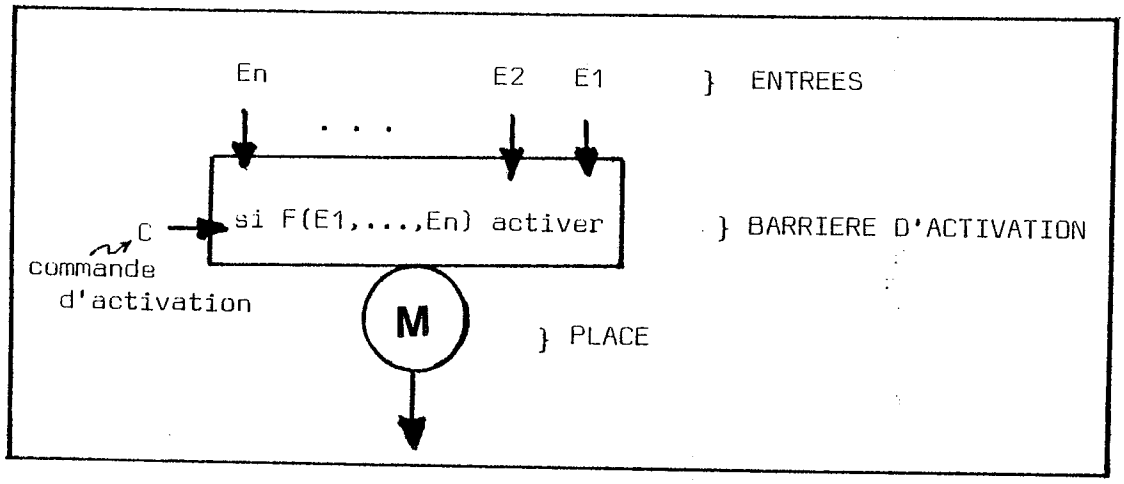


FIG. 2 - ETA synchrone

La barrière d'activation de l'ETA synchrone M assure deux rôles :

- l'évaluation du vecteur d'entrées (E1,...,En) par la fonction booléenne F.
- la validation du chargement de la valeur logique F(E1,...,En) dans la place M se fait par l'entrée C.
  (C) est appelée commande d'activation.

La place M, qui est symbolisée par un cercle sur la Figure 2 , représente un élément de mémorisation.

L'instruction d'affectation associée à l'ETA synchrone M est :

```
<C>   M  <=  F(E1,...,En);
```

ce qui signifie:

Si la commande d'activation (C) est active (C=1)
alors la place M reçoit la valeur logique F(E1,...,En).

2.2. - ETA séquentiel asynchrone

De manière concise, l'ETA séquentiel asynchrone est appelé ETA asynchrone.
Sa représentation graphique est la suivante :

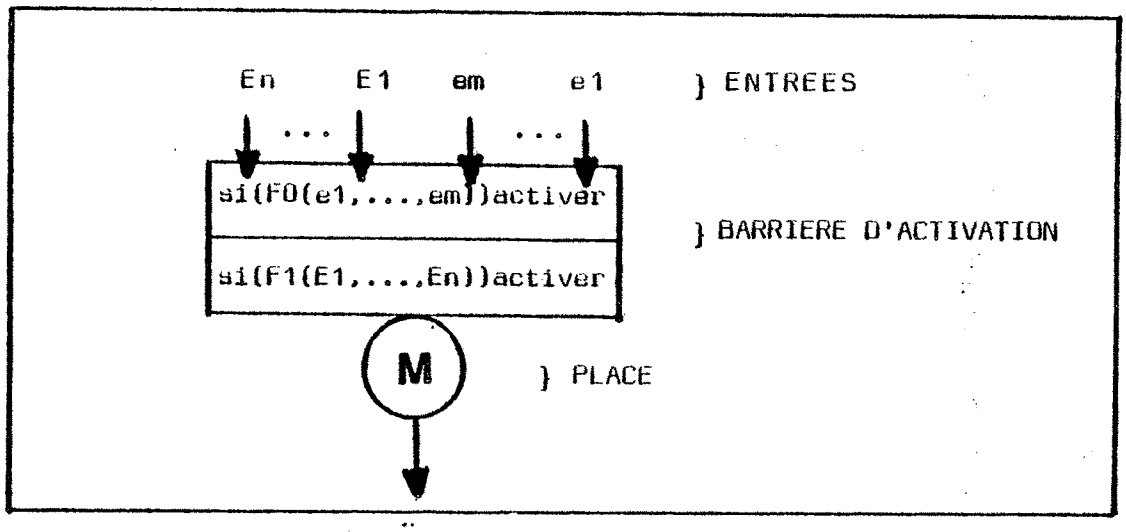


FIG. 3 - ETA asynchrone

La barrière d'activation de l'ETA asynchrone M assure la validation du chargement des valeurs logiques (1,0) dans la place M.

La validation du chargement de la valeur logique (0) se fait par la fonction booléenne FO qui évalue le vecteur d'entrées (e1,...,em). FO est appelée : fonction de mise à zéro.

La validation du chargement de la valeur logique (1) se fait par la fonction booléenne F1 qui évalue le vecteur d'entrées (E1,...,En). F1 est appelée : fonction de mise à un.
La place M qui est symbolisée par un cercle sur la Figure 3, représente un élément de mémorisation.

L'instruction d'affectation associée à l'ETA asynchrone M est :

< FO (e1,...em)> M←0, <F1 (E1, ...En) > M ←1 ;

ce qui signifie
Si la fonction de mise à zéro est active [FO (e1, ..., em) = 1]
alors la place M reçoit la valeur logique (0).
Si la fonction de mise à UN est active [F1 (E1,... En) = 1]
alors la place M reçoit la valeur logique (1).

Remarque :

L'introduction de l'ETA asynchrone dans les primitives de base, ne signifie en rien que le formalisme DELTA est destiné à la description des machines séquentielles asynchrones*.

La priorité d'activation (mise à zéro, mise à UN) dans le cas où les deux fonctions (F0, F1) sont actives en même temps, n'est pas introduite directement dans la définition de l'ETA asynchrone pour conserver à celui-ci un aspect de généralité efficace.

Cependant, ceci peut et doit être pris en compte par les outils de CAO, qui opèrent sur les descriptions DELTA.

---

* voir définition (PARTIE.3)

2.3. - ETA combinatoire

La représentation graphique de l'ETA combinatoire est la suivante :

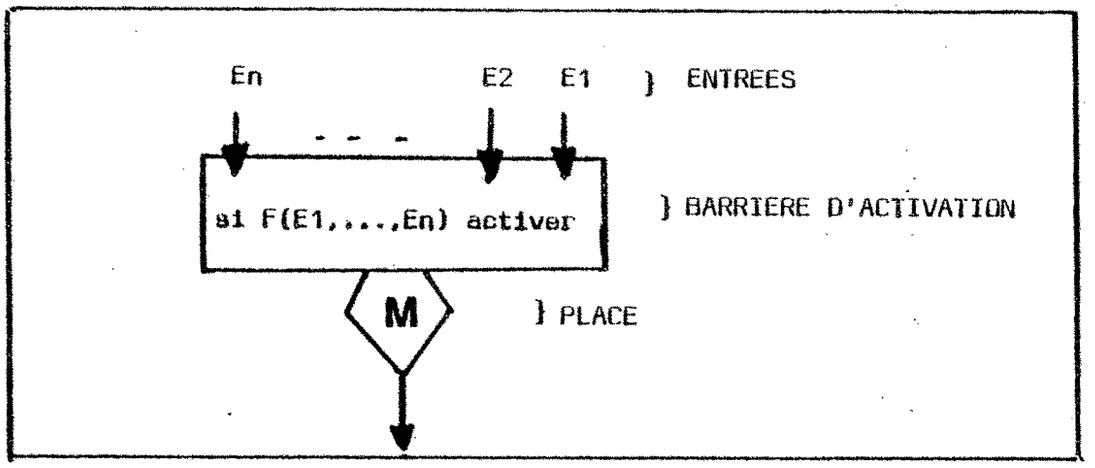


FIG. 4 - ETA combinatoire

La barrière d'activation de l'ETA intermédiaire M assure l'évaluation du vecteur d'entrées (E1,...,En) par la fonction booléenne F et le positionnement de la valeur logique de la place M par simple connexion.

La place M, symbolisée par un pseudo-losange (FIG. 4), représente un élément de connexion.

L'instruction de connexion associée à l'ETA combinatoire M est :

```
M := F(E1,...,En) ;
```

Ce qui signifie:

> Qu'à tout vecteur d'entrée (E1,...,En) la valeur logique de la place M est définie par F(E1,...,En).

3 - Simplifications destinées à alleger le formalisme

Certaines règles de description sont introduites pour faciliter et simplifier la manipulation des primitives de base du formalisme DELTA.

Ceci ne modifie en rien les caractéristiques architecturales et comportementales mise à jour dans la notion d'ETA, mais permet à l'utilisateur d'accéder à tous les détails d'une description DELTA en supprimant les contraintes dues à la notion de boîte noire.

Nous présentons dans ce qui suit, trois règles qui sont destinées à enrichir (encore plus) le formalisme DELTA notamment sur les points suivants :

- faciliter la manipulation des barrières d'activation [SOUPLESSE],
- améliorer l'évaluation des barrières d'activation [OPTIMISATION],
- rendre plus claires les descriptions DELTA [CLARTE].

3.1. - Eclatement des barrières d'activation

Pour différents besoins (optimisation, simulation de pannes ...), l'utilisateur peut vouloir accéder à un détail particulier de la barrière d'activation d'un ETA.

Les barrières d'activation ne doivent pas être considérées comme des boîtes noires et tout opérateur de celles-ci doit être accessible.

Dans l'autre sens, une description DELTA trop détaillée peut gêner parfois le concepteur qui ne souhaite pas s'embarrasser de certains détails.

Ceci se réalise d'une manière très simple en exploitant les propriétés des expressions booléennes (associativité, commutativité ...).

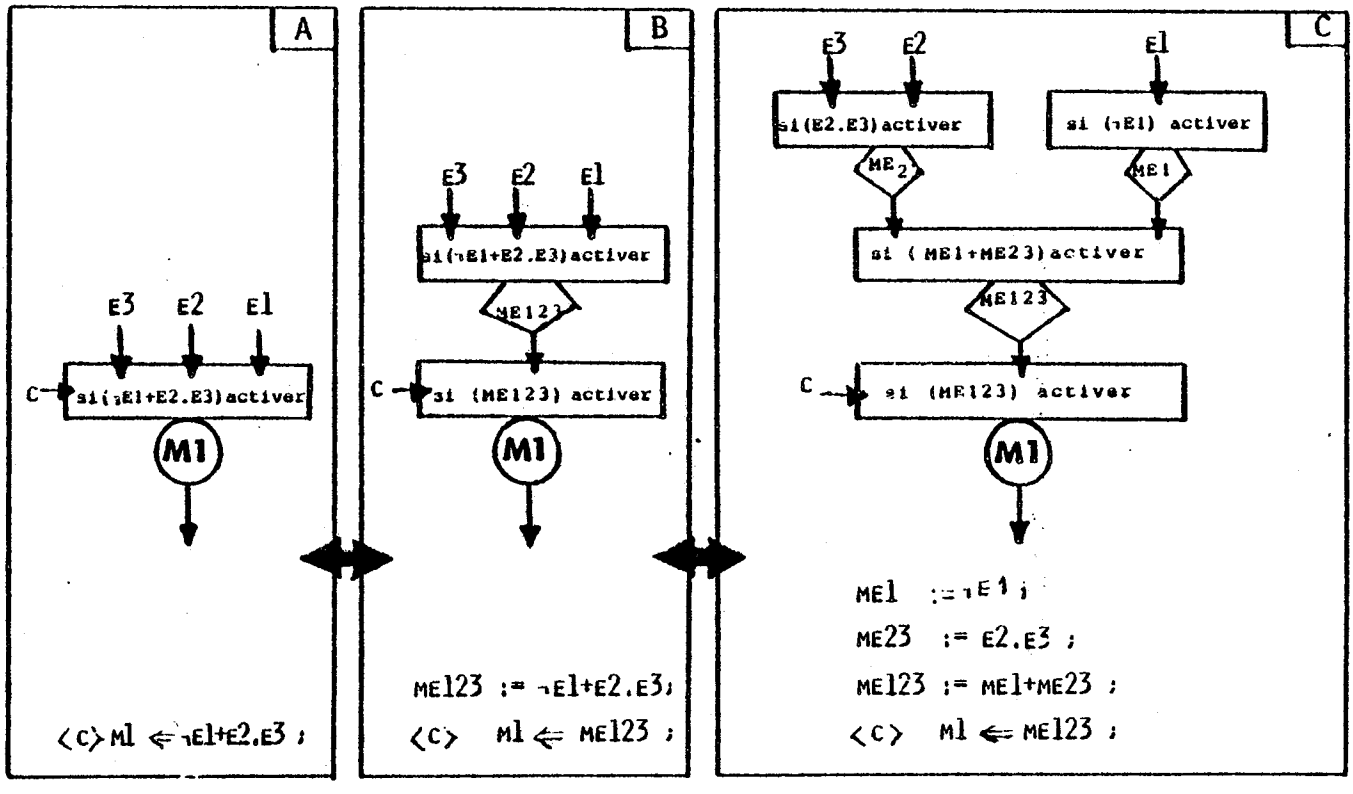


FIG.5 :Différent degré de finesse dans la description d'un ETA.

La description de l'ETA synchrone M1 peut etre realisée de differentes manieres selon le degré de decomposition desire. La figure (5A) represente la description la plus dense alors que celle de la figure (5C) est la plus detaillée (dans le sens de DELTA).

3.2. - ETA Combinatoire du type commande

L'évaluation des expressions booléennes des barrières d'activation pour certains ETA combinatoires peut s'avérer longue et inutile.

Exemple:

Soit à évaluer la valeur logique de l'ETA combinatoire M3,
M3 := C. (M6 + E2. E7. M4 + E9. M2 + M6. M1) ;
Si la variable C est nulle alors M3 prend la valeur logique "0" quelque soit la valeur de l'expression logique (M6 + E2.E7...) qu'on peut ne pas évaluer.
Si, par contre, la variable C est égale à "1" alors M3 prend la valeur de l'expression logique,
(M6 + E2. E7. M4 + E9. M2 + M6. M1) qu'il faut évaluer.

Aussi, nous introduisons l'usage des instructions conditionnelles de connexion dont la forme (ex précédent) est :
si (C = 1) alors M := (M6 + E2. E7. E4 + E9. M2 + M6. M1) sinon M := 0 ;

Nous symbolisons dans le formalisme DELTA, cette instruction conditionnelle de connexion par :

<C> M := (M6 + E2. E7. M4 + E9. M2 + M6.M1) / M := 0 ;

Ceci permet un gain appréciable en temps d'évaluation.
Les ETA combinatoires qui sont décrits par de telles instructions conditionnelles sont dits du type commande. De manière concise, les ETA combinatoires du type commande sont appelés : COMMANDES.
La représentation graphique d'une commande (avec sa description équivalente) est :

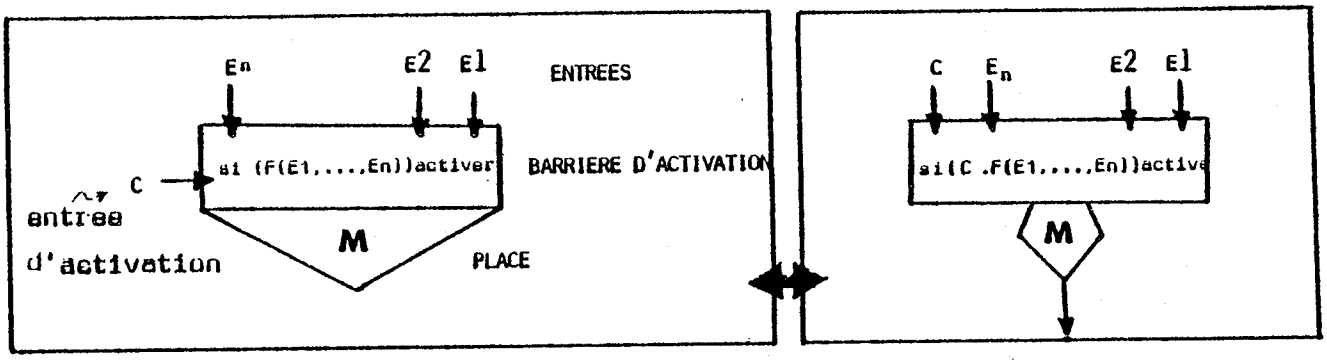


FIG.6A :Representation DELTA d'une Commande

FIG.6B :DELTA Standard

L'instruction conditionnelle de connexion qui est associée à la commande M (Fig. 6A) est : < C > M := F (E1,...En) / M := 0 ;

Ceci signifie :

Si l'entrée C est active (C = 1) alors la place M prend la valeur logique F (E1, ... En) qu'il faut évaluer.

Si l'entrée C n'est pas active (C = 0) alors la place M prend la valeur logique "0" quelque soit la valeur logique de F (E1,..., En) qu'il ne faut pas évaluer.

... L'entree (C) est appelée ENTREE-DE-VALIDATION .

3.3.- Codage des ETA synchrones :

L'utilisateur peut vouloir éviter des détails non indispensables au moment de la réalisation manuelle d'une description DELTA. Une forme de codage de la description des ETA synchrones peut être introduite pour faciliter la tâche de l'utilisateur.

Le regroupement de la description d'un ou plusieurs ETA synchrones avec celle d'un ETA combinatoire est appelé : ETA combinatoire codé.

Ceci peut s'avérer très intéressant quant à l'allégement des descriptions DELTA.

La représentation graphique d'un ETA combinatoire codé (avec sa représentation graphique standard) est :

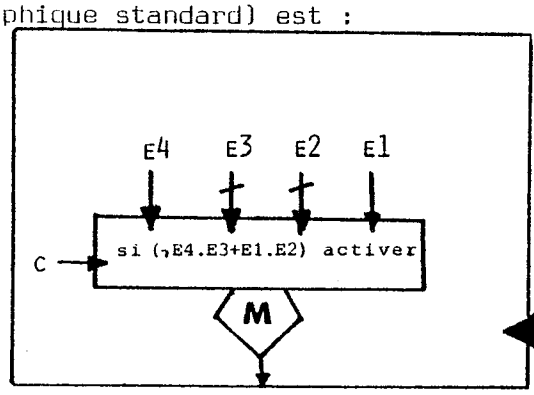


Fig.7A : ETA combinatoire Code

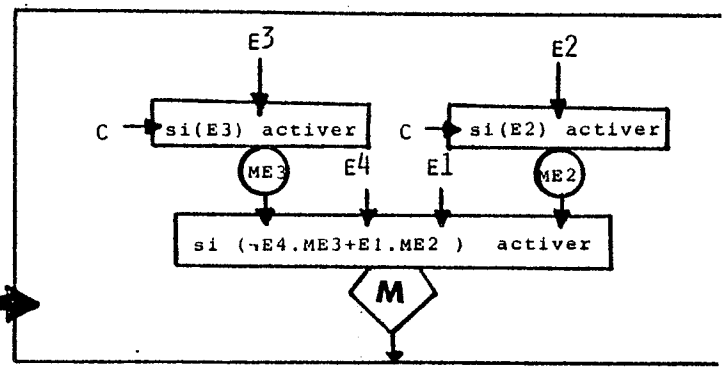


Fig 7B : Description DELTA stand

chaque entrée barrée (Fig 7A) indique que celle-ci se fait à travers un élément de mémorisation dont la commande d'activation est l'entrée C. Les instructions associées à ces descriptions DELTA sont :

- pour l'ETA combinatoire code (Fig. 7A) ;

  M := (¬E4. E3 <C> + E1. E2 <C>) ;

- pour la description DELTA standard équivalente (Fig. 7B) ;

  <c> ME2 ← E2 ;

  <c> ME3 ← E3 ;

  M := (¬E4. ME3 + E1. ME2) ;

pour l'ETA combinatoire codé, ceci signifie :

Si la commande d'activation c est active (c = 1) alors la place M prend la valeur logique ($\neg$E4. E3 + E1.E2).

Si la commande d'activation c n'est pas active (c = 0) alors la place M prend la valeur logique ($\neg$E4. ME3 + E1. ME2) ou (ME1, ME3) indiquent les valeurs mémorisées de (E2, E3) (voir, fig. 7B).

Au niveau interne de l'ordinateur toutes les descriptions DELTA doivent être implantées sous la forme standard (non codée) pour faciliter leur manipulation.

Remarque :

Une critique vient tout de suite à l'esprit après la lecture de cette première partie, qui est :
Le fait de symboliser un élément de mémorisation par un rond (place d'un ETA séquentiel) introduit tout de même une notion de boîte noire dans les primitives de base du formalisme DELTA, et ainsi soulève une contradiction avec le fait d'affirmer que le formalisme DELTA permet l'expression et l'accès à tous les détails d'un objet matériel. En effet, la notion d'ETA séquentiel ne permet pas à l'utilisateur d'accéder directement à la structure matérielle de l'élément de mémorisation (place).

Ceci ne constitue pas une anomalie du formalisme DELTA car le fait de vouloir manipuler la structure matérielle fine d'un élément de mémorisation signifie qu'un choix technologique pour la réalisation a été fait au préalable.

Or, on sait que le formalisme DELTA ne prend pas en compte les paramètres technologiques (du moins d'une manière directe) dans la description des objets matériels.

Ainsi, ce cas particulier de description, loin de déboucher sur une défaillance des primitives de base du formalisme, va nous permettre de montrer la grande souplesse de description de celles-ci. En effet, si la manipulation de la structure matérielle fine de l'élément de mémorisation (place d'un ETA séquentiel devient nécessaire (*), l'utilisateur a toujours la possibilité de réaliser cette description fine par un ensemble d'ETA combinatoires (bouclage combinatoire). Présentée de cette manière, l'utilisateur peut non seulement manipuler la structure fine d'un élément de mémorisation mais peut également choisir toute forme de structure pour sa réalisation.

Ceci constitue une façon d'exprimer (de manière indirecte) certains paramètres technologiques à l'aide des primitives de base du formalisme DELTA, quand cela s'avère nécessaire.

(*) Par exemple pour simuler des pannes au niveau de l'élément de mémorisation.

PARTIE.2

QUELQUES REFLEXIONS SUR DELTA ET LES OUTILS DE CAO

PRESENTATION GENERALE

Le formalisme DELTA étant défini, il faut maintenant déterminer si les notions de base mises en place, permettent effectivement d'atteindre les objectifs plus généraux définis à l'origine, à savoir l'informatisation et l'automatisation des traitements (habituellement manuels) effectués sur les différentes descriptions matérielles.

Pour cela, nous situons le rôle du formalisme DELTA à travers les démarches classiques de la conception des machines séquentielles.

Nous montrons la grande souplesse et les avantages introduits par le formalisme DELTA au niveau du traitement automatique.

Nous mettons l'accent sur les points qui nous paraissent importants pour aboutir à un système d'outils de CAO efficace :

- notion de bibliothèque d'outils de CAO,
- amélioration des mécanismes du traitement au niveau des outils de CAO.

IV- LOCALISATION FONCTIONNELLE DE DELTA

1. - Etapes classiques de description

La conception d'une machine séquentielle destinée au traitement d'un certain type d'information est une opération longue et complexe.

Aussi, la décomposition, en un certain nombre d'étapes successives, de la conception est nécessaire pour mener à bien cette tâche. A chaque étape est associée une description sur laquelle une étude particulière doit être effectuée.

Plusieurs méthodes existent quant à la réalisation de cette décomposition, la METHODE DESCENDANTE, qui est basée sur la construction de couches d'interprétations, en est un exemple typique [39]

Une description macroscopique de la machine via un cahier des charges, permet d'aboutir à une description algorithmique et ceci suivant les différentes techniques d'interprétation.

La démarche inverse qui consiste à aboutir à une description macroscopique à partir d'une description algorithmique détermine la METHODE ASCENDANTE (Figure 8).

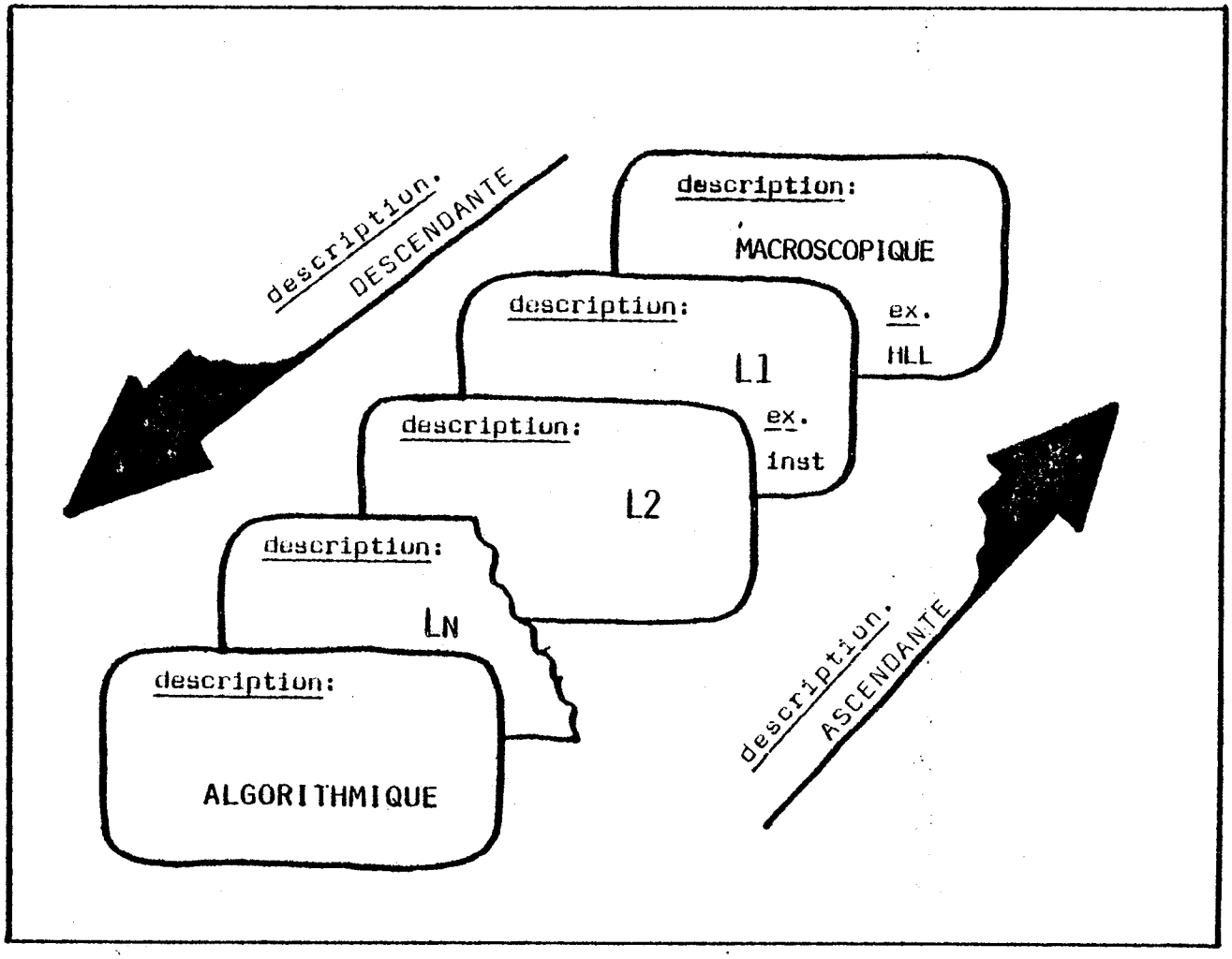


FIG. 8 : Etapes de description comportementales

Notons que ces différentes étapes sont constituées essentiellement par des descriptions comportementales (langages d'interpretation) et donc, aussi bien le passage entre deux étapes successives que l'ensemble des traitements sont automatisables.

Il existe un autre type d'étapes d'interprétation qui est, quant à lui, caractérisé par un ensemble de descriptions matérielles (fig. 9) :

- description LOGIQUE,
- description ELECTRIQUE,
- description TOPOLOGIQUE.

Les techniques d'interprétation de la METHODE DESCENDANTE permettent également de passer d'une description LOGIQUE à une description TOPOLOGIQUE sans trop de problèmes (et réciproquement pour la METHODE DESCENDANTE).

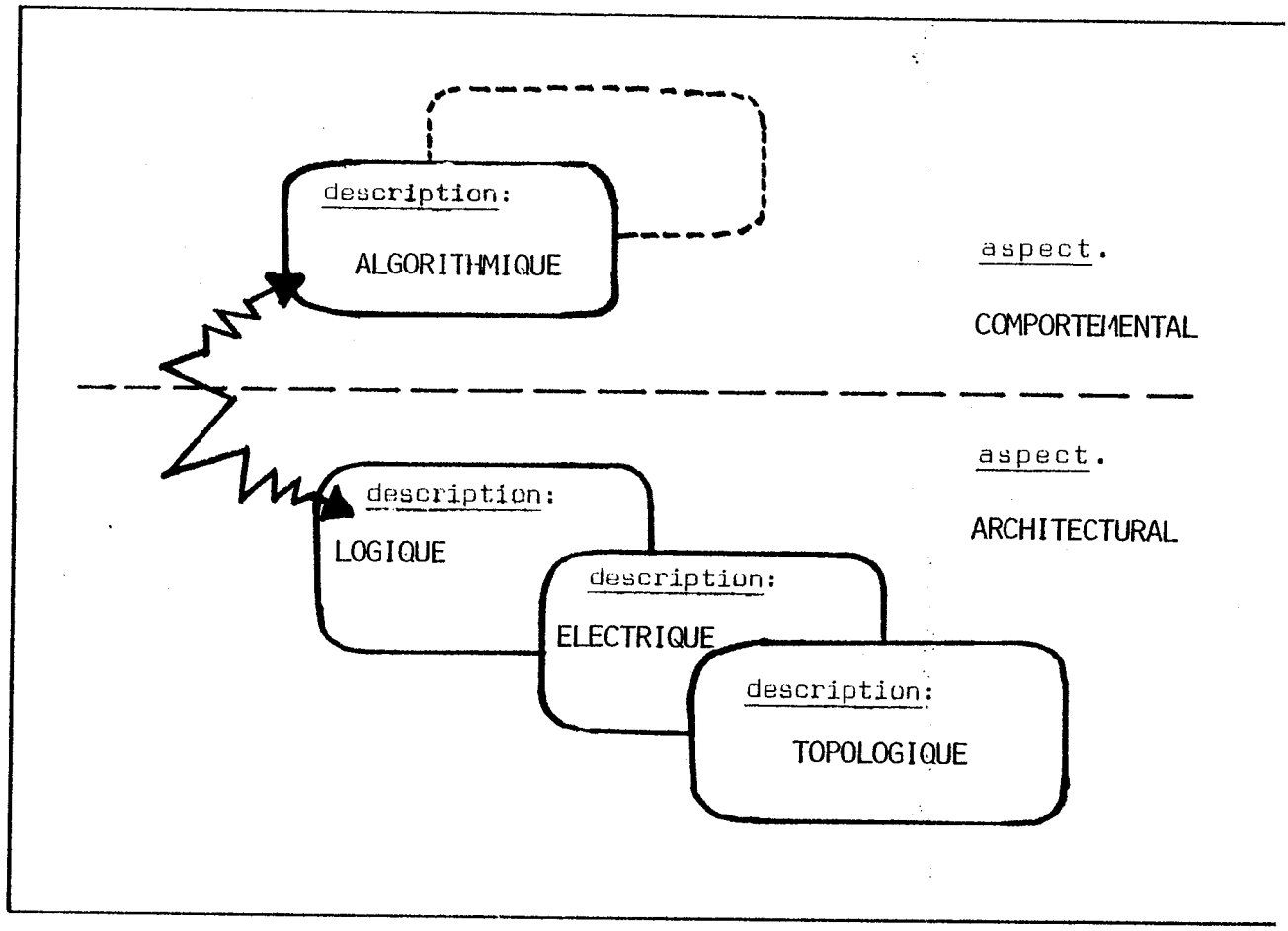


FIG. 9 : Etapes de descriptions architecturales

Lors de la conception d'une machine séquentielle, tous les aspects sont abordés, aussi bien du point de vue du comportement que de celui de l'architecture.
Donc, à un certain stade de la conception, le concepteur est amené à faire le saut entre l'aspect comportemental et l'aspect architectural.

Or, le passage entre les descriptions comportementales et les descriptions ar-

des objets manipulés dans chacunesd'elles. Aussi, il est bien difficile d'informatiser (encore moins d'automatiser) toutes les étapes de la conception d'une machine séquentielle de manière souple et efficace.

Ce que l'on vient de voir dicte la nécessité et les caractéristiques d'une nouvelle étape de la démarche qui sert d'interface à la frontière des deux espaces de descriptions (comportemental, architectural).

La concrétisation de cette nouvelle étape, dans la conception d'une machine séquentielle, peut être réalisée efficacement par le formalisme DELTA qui allie, à travers ses primitives de base, l'aspect comportemental et l'aspect architectural des objets (Fig. 10).

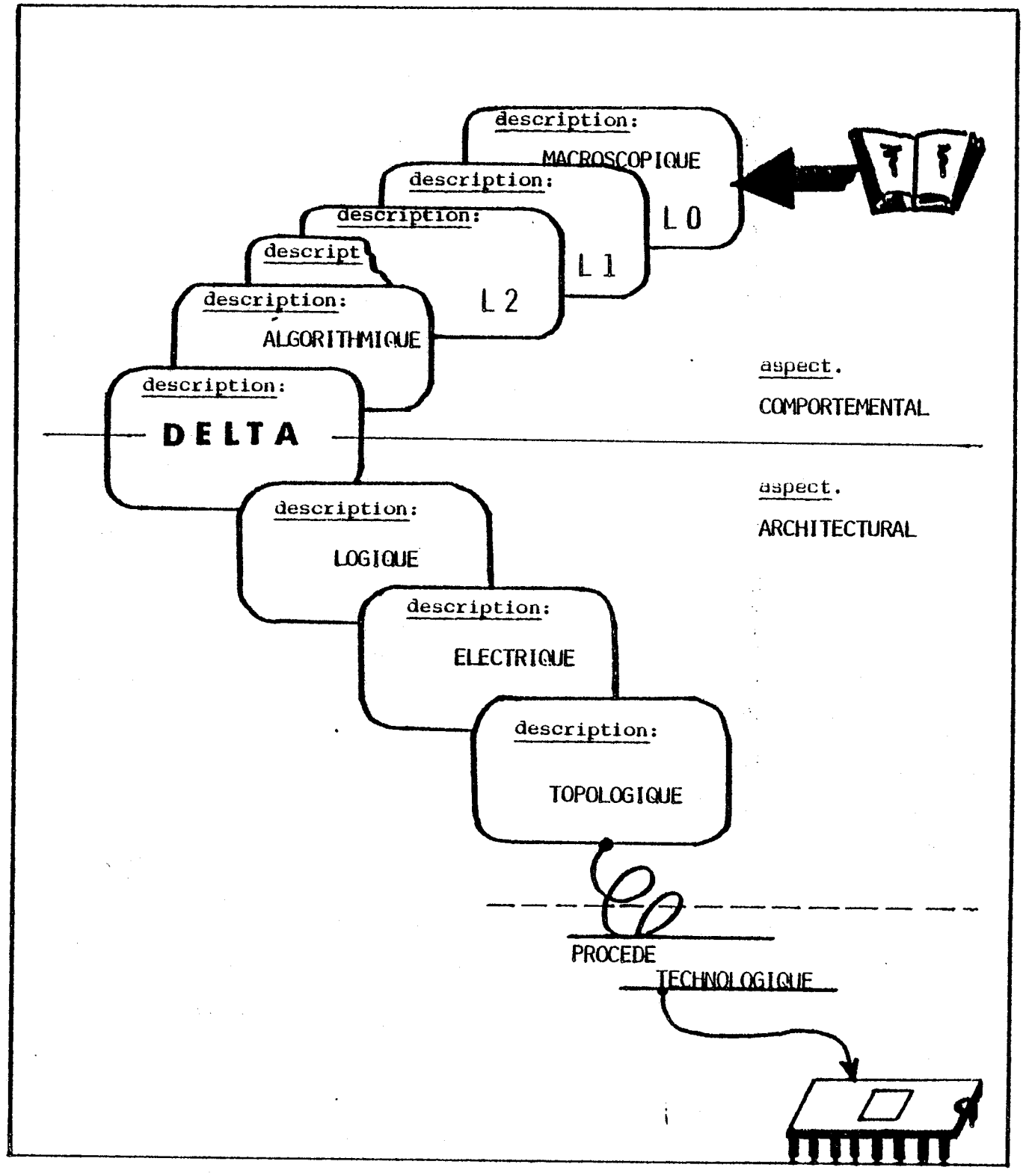


FIG.10 : LOCALISATION DU NIVEAU DES DESCRIPTIONS DELTA

Le formalisme DELTA peut être vu comme un maillon d'une chaîne d'étapes d'interprétation pouvant jouer un rôle privilégié dans la synthese automatique.

Ainsi, le formalisme DELTA, par sa position particulière, assure deux fonctions essentielles pour la synthese automatique :

- informatiser les descriptions architecturales ;
- faciliter le passage entre une description comportementale (ALGORITHMIQUE) et sa représentation architecturale (LOGIQUE).

Ceci est indispensable pour tout traitement automatique efficace, surtout pour les problèmes concernant la description des structures matérielles réalisant le contrôle (*) dans une machine séquentielle.

(*) Partie contrôle d'une machine séquentielle.

2. - Notion de bijection entre DELTA et les objets matériels

Le désir de conserver, au niveau DELTA, une description très proche de celle du matériel vient du fait qu'il est très intéressant de conserver une vision réelle d'un phénomène physique pour étudier efficacement son comportement.

L'un des principaux objectifs du formalisme DELTA est d'allier l'aspect architectural et l'aspect comportemental dans la description des machines sequentielles synchrones.

Aussi, durant toute l'étape de la formalisation, nous avons fait "notre" les points suivants qui nous ont parus importants pour preserver et renforçer l'efficaciter du formalisme DELTA :

- A chaque description DELTA correspond une description matériel d'une manière simple et sans ambiguité (et réciproquement)

BIJECTION

- A chaque description DELTA correspond une description d'une manière simple et directe (et réciproquement).

SOUPLESSE

- Aucune contrainte purement technologique ou électrique ne doit apparaître au niveau des primitives de base du formalisme DELTA.

EFFICACITE

- Les descriptions DELTA doivent être relativement indépendantes entre elles et facilement manipulables.

SEGMENTATION

Ceci nous a conduit à établir une correspondance "bijective" (UN à UN) entre les notions de base du formalisme DELTA et les objets matériels de base dans un circuit séquentiels synchrone :

- A chaque opérateur booléen d'une barrière d'activation correspond une porte logique (et réciproquement).

- A chaque place d'un ETA combinatoire correspond un élément de connexion (et réciproquement).

- A chaque place d'un ETA séquentiel correspond un élément de mémorisation (et réciproquement).

- A chaque commande d'activation d'un ETA synchrone correspond un interrupteur (et réciproquement).

Cette notion de bijection ne sous-entend, en aucune manière, une forme de restriction quant aux différentes réalisation technologiques, que l'on peut faire à partir d'une même description DELTA (*)

Au contraire, cette notion de bijection introduit un assouplissement important et une grande facilité dans le passage entre une description DELTA et sa structure matérielle.

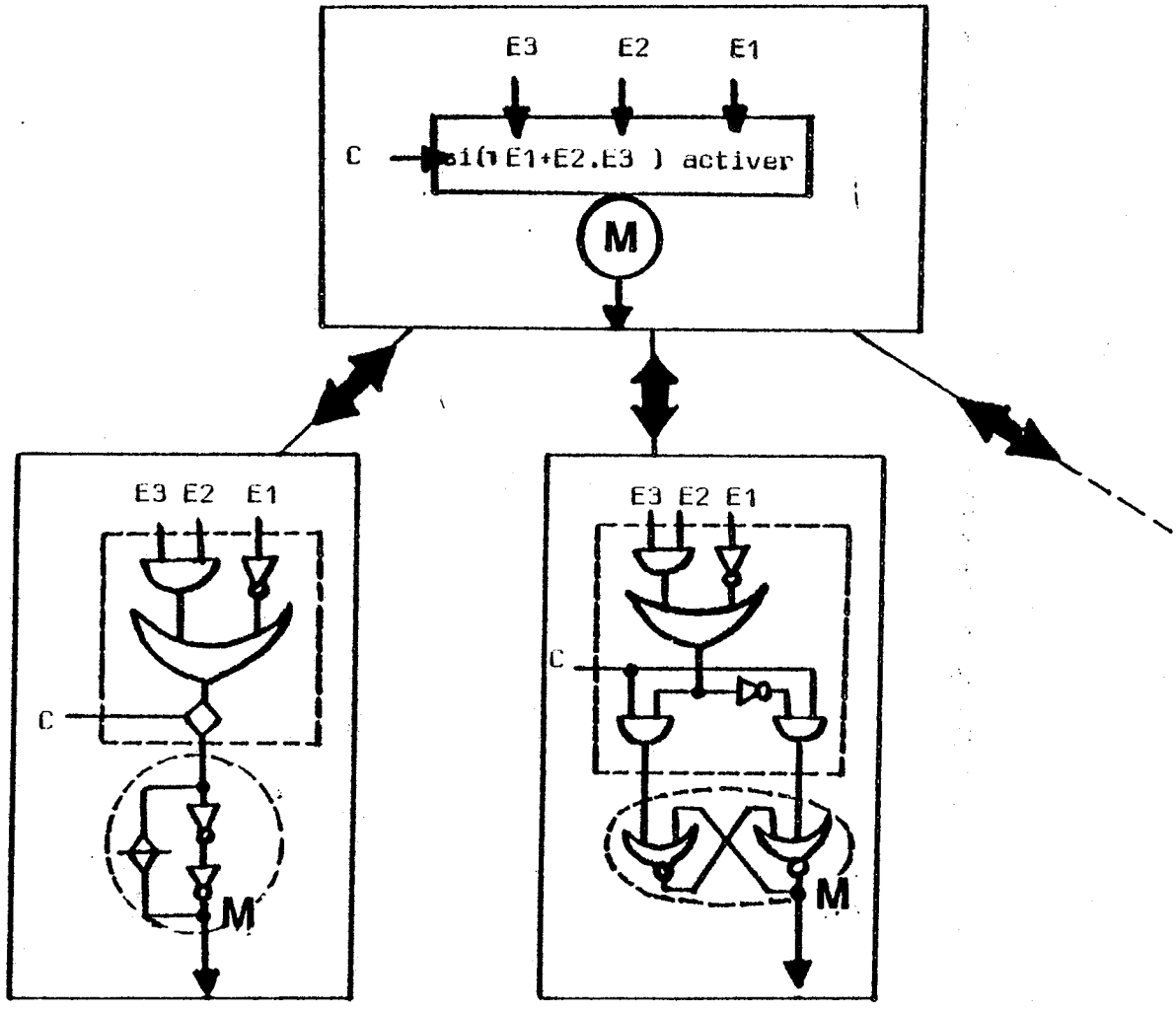
En effet, les primitives de base du formalisme DELTA assurent le passage direct et sans ambiguité entre une description DELTA et sa représentation matérielle (logique, électrique, topologique).

(*) voir PARTIE.3

Comme représentation matérielle, nous choisissons le niveau "logique" pour visualiser le passage entre une description DELTA et sa représentation matérielle (nous aurions pu choisir le niveau électrique ou le niveau topologique, mais ceci revient exactement au même).

Le passage entre ces deux descriptions, se fait de la manière la plus simple et ceci quelque soit la technologie utilisée pour la réalisation.

Exemple 1 : Cas d'un ETA synchrone.



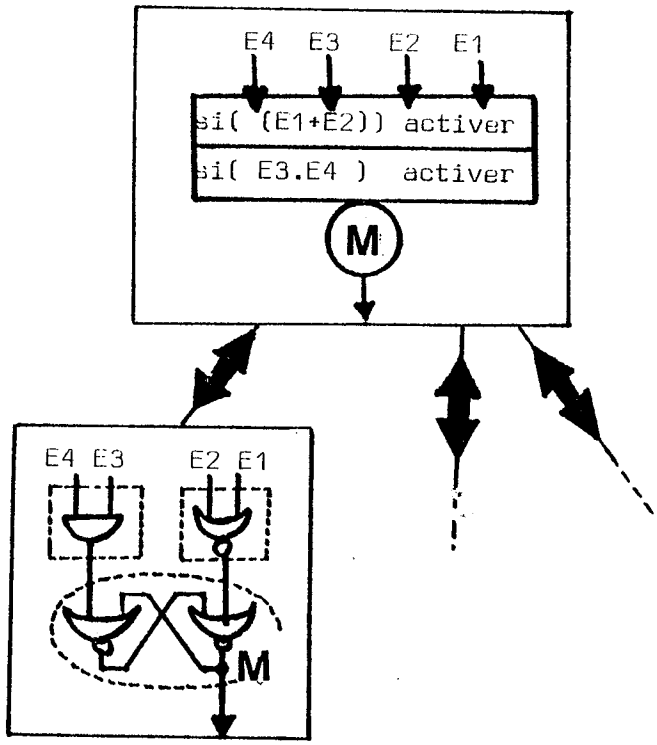
technologie MOS

technologie TTL

...autres

Exemple 2 :

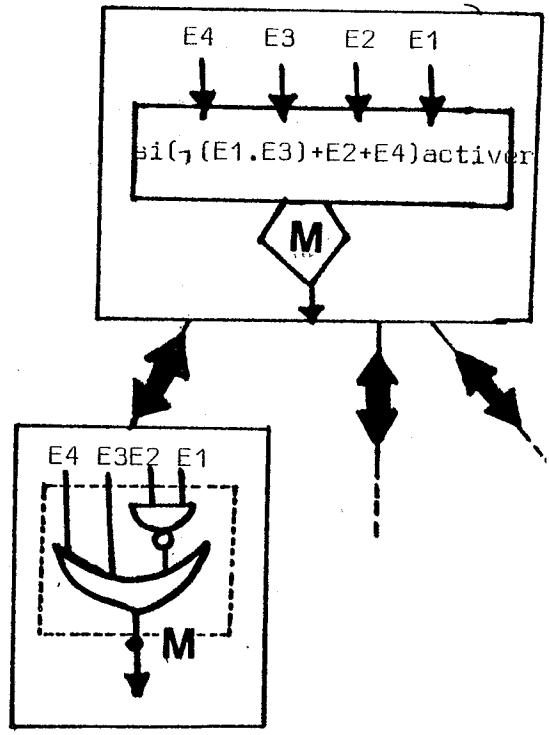
Cas d'un ETA synchrone



Technologie MOS ...Autres

Exemple 3 :

Cas d'un ETA combinatoire

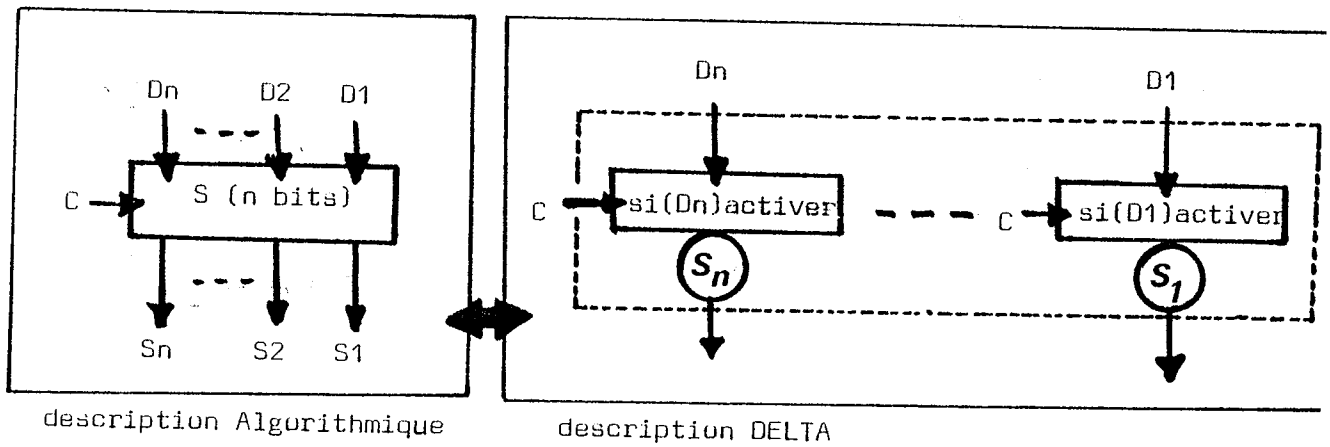


Technologie MOS ...Autres

Le passage entre une description DELTA et une description algorithmique (contrôle/opérateur) se fait d'une manière souple et facile.(Voir exemple suivant).

Exemple 4 :

Cas d'un registre S à n bits (représentation graphique)

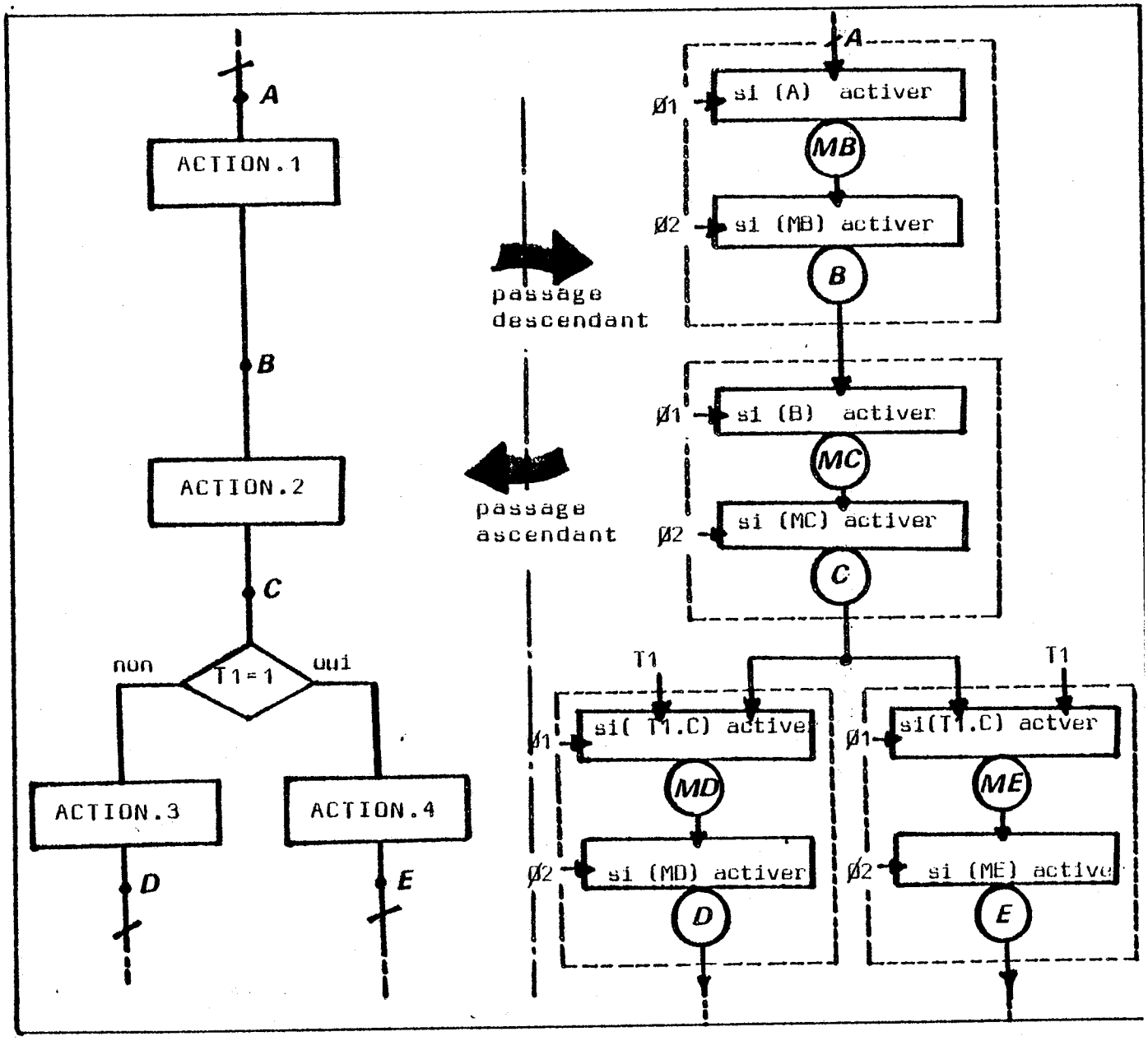


description Algorithmique description DELTA

Il en est de même pour les autres types d'opérateurs (additionneur , multiplexeur , bascule, ...) ainsi que pour les algorithmes de contrôle.

Exemple 5 :

Nous présentons le passage entre une description algorithmique de contrôle et la description DELTA du séquenceur de la partie contrôle (pour une question de clarté, la description DELTA de l'interface de commande, n'a pas été présentée) (*). Nous choisissons une machine séquentielle biphase (*).



organigramme de controle description DELTA du srquenceur

(*) : Voir PARTIE.3

3 - Paramètres technologiques et description DELTA

Le formalisme DELTA établit un pont efficace entre l'aspect logiciel et l'aspect matériel d'une machine séquentielle synchrone.

Les primitives de base du formalisme DELTA permettent un passage quasi automatique entre une description algorithmique et une description DELTA et ceci par la définition de simples règles de correspondance, entre les éléments de ces deux niveaux de description (voir paragraphe precedent).

Nous savons, que pour une question d'efficacité de description, les paramètres technologiques de réalisation n'ont pas été introduits au niveau des primitives de base du formalisme. Aussi, il serait intéressant de voir comment s'opère la prise en compte de ces paramètres technologiques.

Au moment de la réalisation physique de la machine, un certain nombre de décisions doivent être prises quant aux moyens à utiliser pour la mise en oeuvre de cette réalisation :

- choix de la technologie à utiliser,
- choix de la structure des éléments de mémorisation,
- ... et bien d'autres, selon les caractéristiques de la technologie utilisée.

La notion de bijection introduite dans le formalisme DELTA permet d'effectuer un passage simple et direct entre une description DELTA et une description matérielle (logique, électrique ou topologique) et ceci par de simples règles de correspondance (voir paragraphe précédent).

Ces règles de passage doivent exprimer la prise en compte de tel ou tel paramètre technologique.

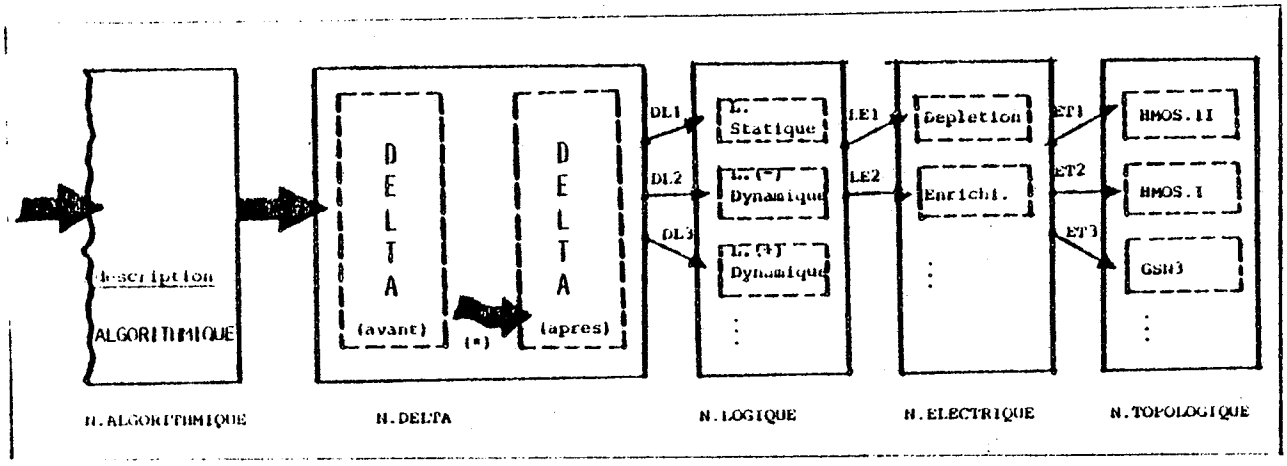


FIG. 11 : Paramètres technologiques pour la technologie MOS.

Pour une technologie MOS, les paramètres technologiques sont introduits progressivement à travers les règles de passage (méthode descendante) entre les différents niveaux de description :
par exemple figure 11. :

PASSAGE ENTRE LES DESCRIPTIONS DELTA α LOGIQUE

DL1 = élément de mémorisation statique,

DL2 = élément de mémorisation dynamique avec une logique positive/partie contrôle (**)

DL3 = élément de mémorisation dynamique avec une logique alternée (**) (positive, négative) / partie contrôle

.
.
.
et d'autres

(*) voir paragraphe suivant

(**) voir partie 3

PASSAGE ENTRE LES DESCRIPTIONS LOGIQUES α ELECTRIQUES

LE1 = MOS de charge à déplétion (canal N),

LE2 = MOS de charge à enrichissement (canal N),

.
.
.

et d'autres.

PASSAGE ENTRE LES DESCRIPTIONS ELECTRIQUES α TOPOLOGIQUES

ET1 = réalisation des masques en HMOS.II

ET2 = réalisation des masques en HMOS.I

ET3 = réalisation des masques en GSN3 (6n),

.
.
.

et d'autres.

Cette décomposition est souhaitable dans le cas où on veut conserver la trace d'une description d'un certain niveau.

Ainsi, pour un même projet, on peut vouloir effectuer plusieurs réalisations technologiques.

Cependant, cette décomposition ne doit pas être considérée comme une contrainte, car des règles permettant le passage direct entre une description DELTA et une description électrique ou topologique, peuvent être définies

d'une manière facile grâce à la notion de bijection introduite dans le formalisme.

De telles règles de passage peuvent s'avérer très intéressantes surtout pour les techniques de réalisation basées sur la notion de bibliothèque de briques topologiques [30].

V - QUELQUES REFLEXIONS SUR L'ASPECT CAO

1 - Structures d'implantation sur Ordinateur des descriptions DELTA :

Les structures internes pour l'implantation des descriptions DELTA doivent non seulement restituer les différentes caractéristiques des primitives de base du formalisme, mais elles doivent également permettre une manipulation souple et efficace par les outils de CAO.

Les caractéristiques des ETA sont prises en compte, au niveau des mécanismes du traitement automatique.

Une description DELTA peut être considérée comme un réseau dont les noeuds seraient les ETA.

Aussi, pour une implantation sur ordinateur, un ETA quelconque du formalisme DELTA peut être représenté par la structure de données suivante :

Structure générale d'implantation sur ordinateur des descriptions DELTA.

dans laquelle :

NOM ,

est un identificateur qui permet de reconnaître l'ETA décrit.

VALEUR ,

est une variable booléenne qui indique la valeur d'activation de l'ETA.

TYPE ,

est une variable générale qui indique la nature de l'ETA.

Exemple :

TYPE = 0 pour un ETA synchrone,
= 1 pour un ETA asynchrone,
= 2 pour un ETA combinatoire,
= 3 pour une commande.

BARRIERE,

est une variable qui pointe sur les éléments de la barrière d'activation de l'ETA.

SUCCESSEURS,

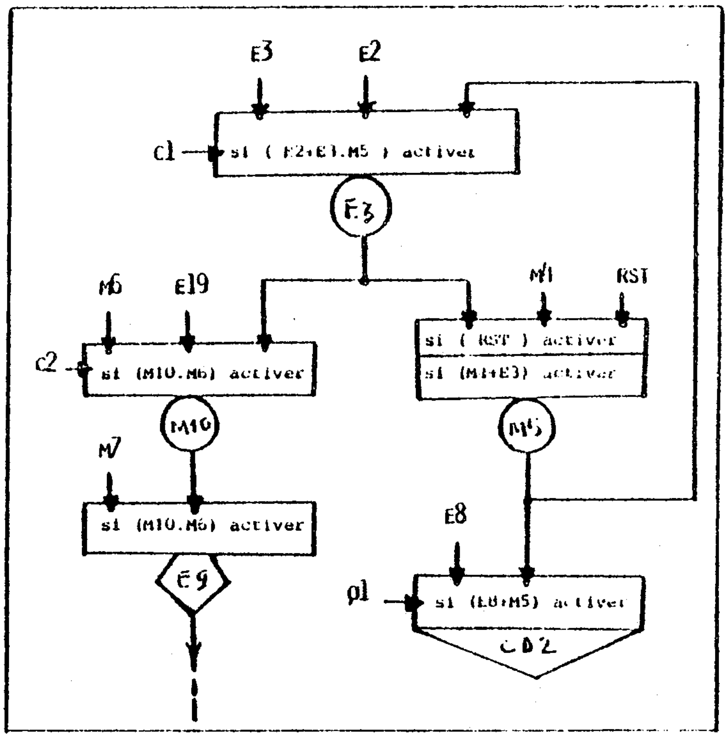
est une variable qui pointe sur l'ensemble des ETA successeurs .

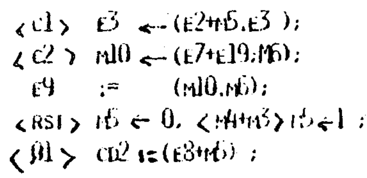
(Un ETA M est dit successeur d'un ETA E si E intervient dans l'expression booléenne de la barrière d'activation de l'ETA M).

Notons que le nombre d'informations servant à la représentation interne d'un ETA peut être augmenté, et ceci d'une manière progressive, selon les spécifications de tel ou tel traitement.

Dans l'exemple suivant, nous présentons une description DELTA et sa représentation interne. Pour une question de commodité, nous avons utilisé des structures de listes pour représenter l'ensemble des successeurs et les barrières d'activation des ETA.

Exemple: Suit la description DELTA suivante.

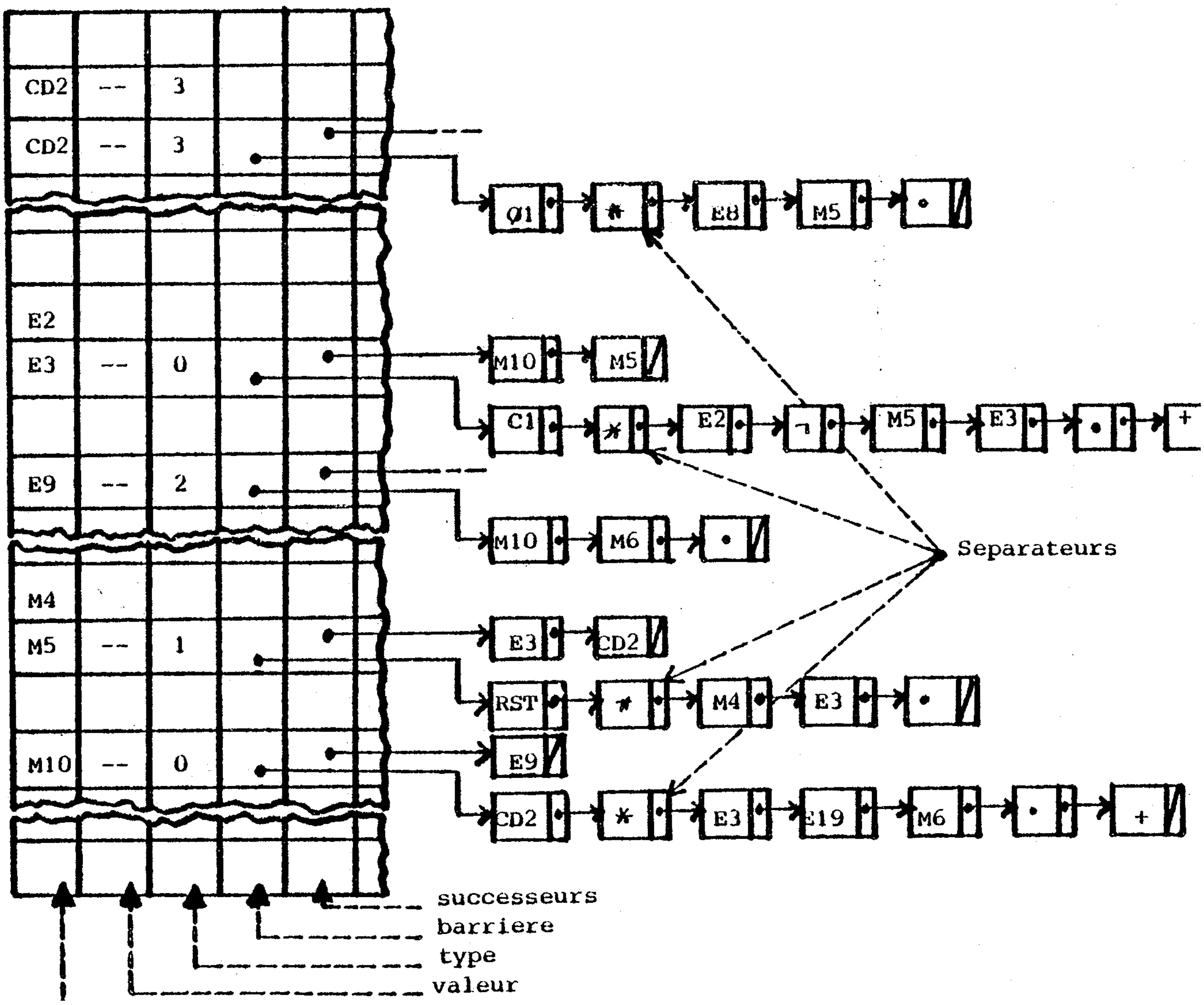




REPRESENTATION GRAPHIQUE

REPRESENTATION LOGICIELLE

La representation de cette description DELTA au niveau interne de la machine informatique est :



Remarque :

L'implantation interne des descriptions DELTA est réalisée par un outil de CAO spécialisé (ex:compilateur).

Cet outil de chargement doit effectuer le décodage des ETA synchrones si cette forme de description a été utilisée (voir PARTIE 1) , ainsi que toute autre forme de manipulation telle que :

- éclatement des barrières d'activation, (par la création de nouveaux ETA),
- suppression de certains ETA, ...

Pour améliorer le traitement des barrières d'activation, les expressions booléennes peuvent être implantées sous la forme post-fixée (comme dans l'exemple précédent).

2. - Notion de bibliothèque d'outils de CAO

Dans les démarches classiques de la conception manuelle des machines séquentielles, les risques d'erreurs, dûs à l'intervention humaine, obligent le concepteur à répéter certains traitements à chaque étape de la conception (parexemple, la vérification de la validité des descriptions dans chaque niveau).

Un rôle supplémentaire que DELTA peut assurer est celui d'éviter la répétition de ces traitements (répétition indispensable dans le cas d'une conception non automatique).

Il suffit pour cela de regrouper l'ensemble des traitements dans le niveau DELTA :

- simulation du fonctionnement,
- organisation de l'architecture,
- optimisation,
- simulation de pannes, ...

De plus, la réalisation de ces différents traitements dans le niveau DELTA a l'avantage d'aboutir à un ensemble d'outils de CAO compatibles entre eux (car opérant sur les mêmes types de données: les descriptions DELTA).

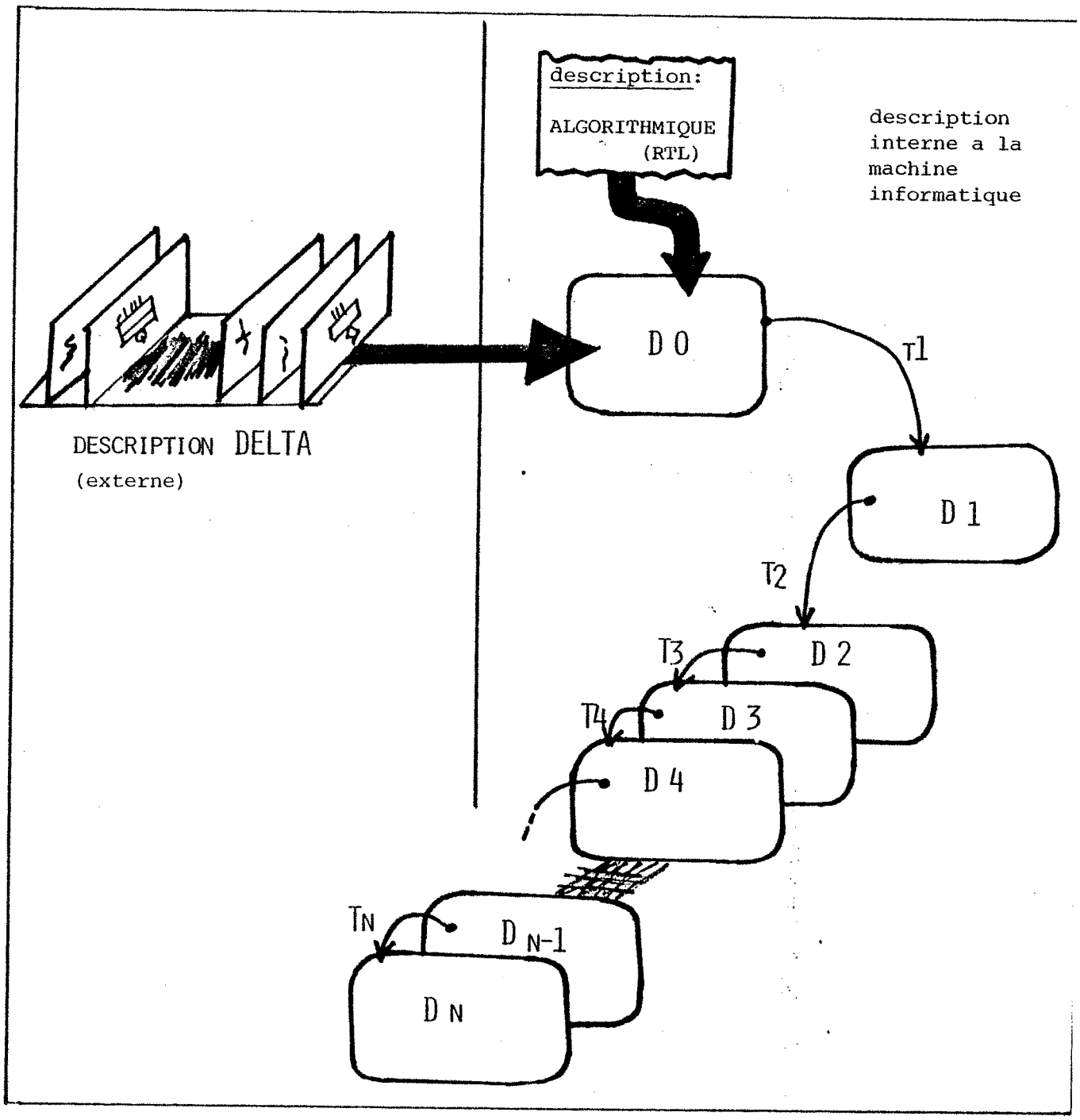


FIG. 12 : Organisation du traitement automatique dans le niveau DELTA.

Un traitement automatique souple et efficace peut se faire en plusieurs étapes. Ainsi, dans le niveau DELTA, chaque outil de CAO (Ti) opère sur une description DELTA (Δi-1) pour réaliser un traitement particulier en générant une nouvelle description DELTA (Δi) sur laquelle opère un autre outil de CAO (T+1) qui génère à son tour une nouvelle description DELTA (Δi+1) ...

La description DELTA initiale (Δ0) peut être obtenue de plusieurs manières, entre autres :

- à partir d'une description algorithmique interne (exemple: type RTL),
- à partir d'une description DELTA externe réalisée manuellement par l'utilisateur.

La dernière description DELTA (Δn), issue de la description (Δ0) après la réalisation des différents traitements dans le niveau DELTA, est l'image fidèle de la structure matérielle servant à la réalisation physique du circuit intégré.

Exemple :

Soit une description DELTA (Δ0) issue d'un algorithme de contrôle* contenant éventuellement des erreurs de conception. Un outil de CAO (T1) qui réalise la simulation logique opère sur (Δ0) en localisant et en permettant à l'utilisateur de corriger l'ensemble des erreurs. Le traitement par (T1) permet de générer une description (Δ1) dépourvue d'erreur sur laquelle opère un outil de CAO (T2) qui réalise l'optimisation des expressions booléennes des barrières d'activation de (Δ1) pour aboutir à une description (Δ2). Ensuite, un outil (T3) permettant l'organisation des ETA séquentiels et des ETA combinatoires de la description (Δ2) permet d'aboutir à une description (Δ3) dont l'organisation architecturale est celle d'une structure micro-programmée ,à base de PLA ou autres ...

---

* La même opération peut être réalisée pour une partie

Cette organisation du traitement dans le niveau DELTA, facilite (énormément) la création d'une bibliothèque d'outils de CAO.

De plus, le choix d'une structure modulaire au niveau des programmes facilite(considérablement) la conception et la réalisation des outils de CAO et permet une grande optimisation quant au volume total du logiciel.

3.- Formalisme DELTA et stratégie du traitement automatique

Le formalisme DELTA qui, comme on vient de le voir, facilite la mise en oeuvre d'une bibliothèque d'outils de CAO, mais il permet également d'améliorer la stratégie des différents traitements automatiques.

En effet, la prise en compte des caractéristiques matérielles et l'absence de paramètres technologiques au niveau des primitives de base du formalisme DELTA permettent d'améliorer (considérablement) les performances du traitement automatique.

L'inconvénient majeur de la plupart des outils de description des machines séquentielles dans le niveau logique (LDL, CDL,...), est,qu'ils obligent le concepteur à prendre en considération certains paramètres technologiques sans que cela soit nécessaire.

Ceci alourdit inutilement les différents traitements que l'on peut réaliser dans le niveau logique.

Par exemple, la description d'un élément de mémorisation ou celle d'un interrupteur en termes de portes logiques, ne fait qu'alourdir les mécanismes du traitement automatique (problème du bouclage d'un élément de mémorisation) et allonger le temps de passage dans l'ordinateur.
Soient les descriptions logiques suivantes :

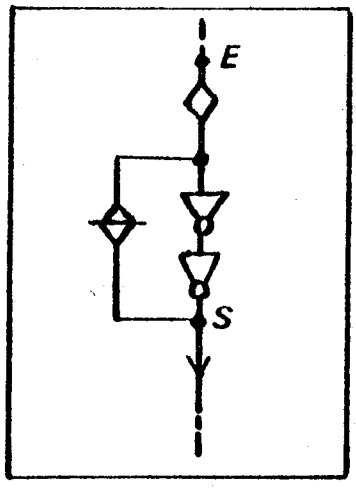


FIG.13A

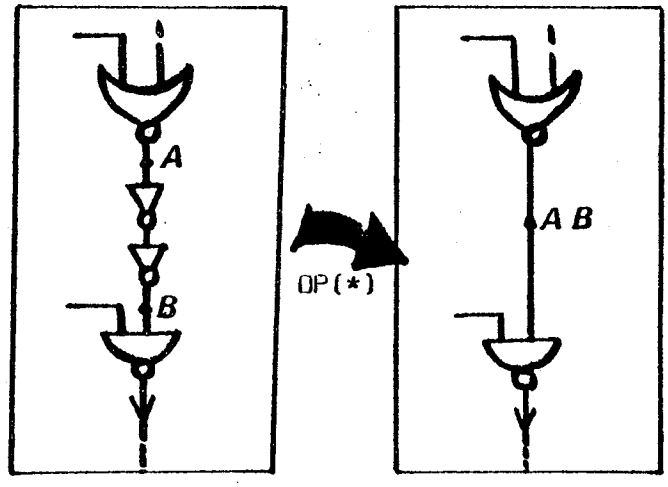


FIG.13B FIG.13C

(*) Passage après optimisation

Les deux inverseurs de la figure (13A) appartiennent à un élément de mémorisation.

Les deux inverseurs de la figure (13B) appartiennent à un circuit combinatoire. Pendant un traitement d'optimisation, les deux inverseurs de la figure (13B) peuvent être supprimer (ce qui se fait dans la plupart des cas) car ils correspondent à la fonction identite (Fig. 13C).

Ceci ne peut pas être fait pour les deux inverseurs de la figure (13A) sous peine de détruire l'élément de mémorisation statique. Donc, pendant l'optimisation d'un schéma logique, il faut faire un traitement particulier pour distinguer ces deux cas de figure (donc complication du traitement).

Le formalisme DELTA permet de supprimer ces inconvénients grâce aux notions de place et de barrière d'activation.

De plus, l'absence de paramètres technologiques au niveau des primitives de base du formalisme DELTA simplifie (énormément) les différents traitements et permet d'introduire plus efficacement, au niveau des mécanismes des outils de CAO, les différentes techniques du traitement manuel du concepteur.

Exemple :

Soit un outil de simulation logique (SD) qui opère sur des descriptions DELTA en exploitant toutes les caractéristiques matérielles mise à jour dans le formalisme DELTA.
Nous considérons, pour cet exemple, la simulation de la partie contrôle d'un circuit séquentiel* biphase. La simulation se fait phase par phase.

---

* Voir définition PARTIE.3

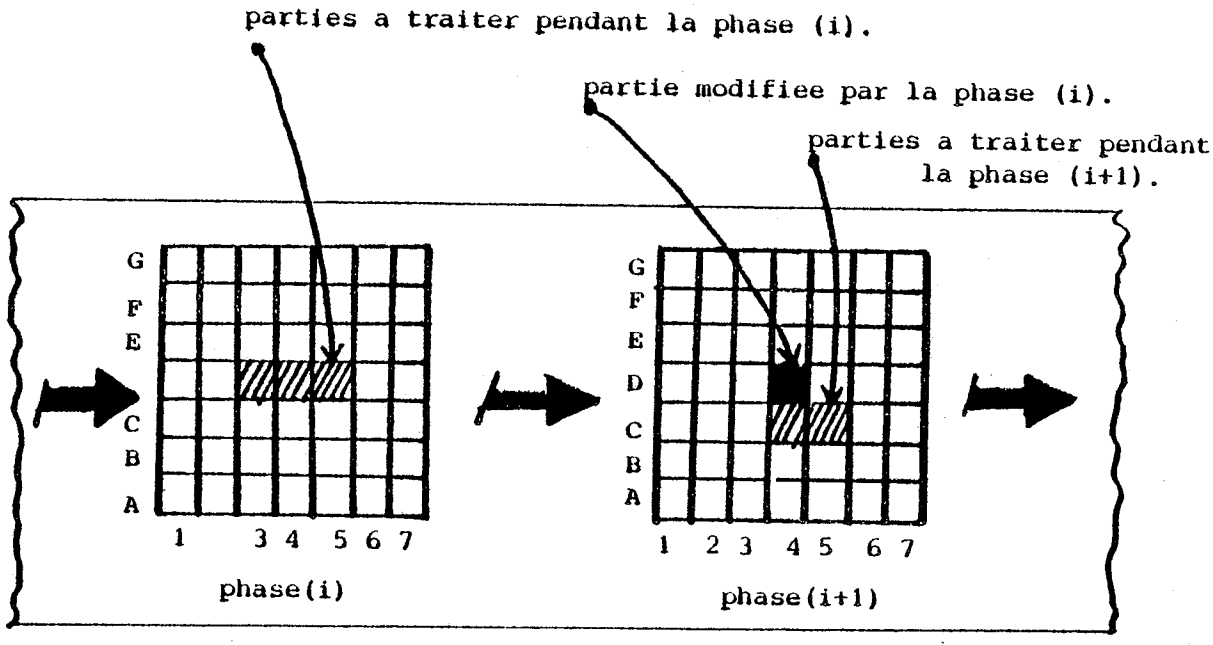


Fig. 14 : trace générale de la simulation par (S Δ)

Le simulateur (S Δ) qui opère sur des descriptions DELTA est dit sélectif car le traitement de la simulation ne porte que sur les parties de la description qui sont susceptibles de modifier l'état général du circuit simulé (Fig. 17).

Cette stratégie de la simulation est liée au fait que dans un circuit séquentiel à un instant donné, seule une partie du circuit est active :

- seuls quelques opérateurs de la partie opérative* sont sollicités à un instant donné,
- seule une U-instruction dans la partie controle* (structure microprogrammée) est activée,
- ...

Ceci est dû aux différentes caractéristiques (Architecturales, comportementales) mises à jour dans le formalisme DELTA qui permet également d'étendre la synthese automatique à des aspects autres que celui de la simple simulation logique.

---

* voir PARTIE.3

PARTIE.3

CARACTERISTIQUES GENERALES DES CIRCUITS SEQUENTIELS

Présentation générale

Le but de cette partie est de présenter les caractéristiques principales des machines séquentielles synchrones. Nous mettons l'accent sur les caractéristiques et les notions matérielles qui nous ont permis de déterminer les primitives de base du formalisme DELTA.

Nous présentons les mécanismes généraux des différents modèles d'architecture qui sont utilisés dans la conception des circuits séquentiels (10) (18) (30) (33), notamment pour les structures suivantes:

- structures câblées,
- structures microprogrammées,
- structures à PLA,
- structures mixtes.

Une approche de formalisation pour une description hiérarchisée des circuits séquentiels est abordée.

VI- CARACTERISTIQUES GENERALES DES MACHINES SEQUENTIELLES

1. - Concepts de base

1.1. - Machine séquentielle

Une machine (*) séquentielle est définie par un 6-uplet (1)

$$\langle Q, q0, E, S, \delta, \gamma \rangle$$

dans lequel :

- $Q = \{q0, q1, \ldots, qn\}$
  est l'ensemble fini de ses etats internes.

- $q0$,
  est l'état initial à l'origine des temps.

- $E = \{e1, \ldots, en\}$
  est l'ensemble fini des vecteurs d'entree que peut recevoir cette machine. Les composantes des vecteurs sont appelées:entrées.

- $S = \{s1, \ldots, sn\}$
  est l'ensemble fini des vecteurs de sortie que peut émettre cette machine. Les composantes des vecteurs

- δ ,
  est la fonction de transition entre les états internes. Elle définit l'état $Q[t+1]$ à l'instant $t+1$, à partir de l'état $Q[t]$ et du vecteur d'entrée $e[t]$ à l'instant t :

$$Q[t+1] = \delta(Q[t], e[t]).$$

(*) Dans tout ce qui suit, nous utilisons indifféremment les termes de machine ou circuit.

- γ ,

est la fonction de sortie dont les caractéristiques déterminent deux familles de machines séquentielles :

Les machines séquentielles de Mealy dont la fonction de sortie γ définit le vecteur de sortie s[t] à partir de l'état interne Q[t] et du vecteur d'entrée e[t] à l'instant t, $s[t] = \gamma(Q[t], e[t])$.

Les machines séquentielles de Moore dont la fonction de sortie définit le vecteur de sortie s [t] à partir de l'état interne Q(t) à l'instant (t), $s[t] = \gamma(Q[t])$.

De cette definition,il resulte que les instants auxquels une machine sequentielle fait evoluer son etat interne ,constituent une suite infinie d'instants:

$$t0, t1, \ldots, t_n, \ldots$$

Suivant les caracteristiques de suite d'instants une machine sequentielle est appelée Synchrone ou Asynchrone:

-Une machine sequentielle est dite SYNCHRONE si les instants auxquels la machine fait evoluer son etat interne constituent une suite infinie denombrable.
Dans ce cas, la suite des instants (t0, t1, ..., t_n, ...) est determinée par l'evolution d'une entrée (appelée generalement HORLOGE) de la machine.

-Une machine sequentielle est dite ASYNCHRONE si les instants auxquels la machine fait evoluer son etat interne constituent une suite infinie non denombrable

1.2. Circuit combinatoire

Un circuit combinatoire est un 3-uplet

$$<E, S, \beta>$$

dans lequel :

$E = \{e1, \ldots, en\}$
est l'ensemble fini des vecteurs d'entrée que peut recevoir ce circuit.

$S = \{s1, \ldots, sn\}$
est l'ensemble fini des vecteurs de sortie que peut émettre ce circuit.

β,
est la fonction de sortie ; elle définit, à tout instant t, le vecteur de sortie $s[t]$ à partir du vecteur d'entrée $e[t]$:
$s[t] = \beta(E[t])$.

De cette définition, il résulte que les instants auxquels un circuit combinatoire fait évoluer ses vecteurs de sortie constituent une suite infinie non dénombrable.

1.3. Notion de partie opérative et de partie contrôle

Les machines séquentielles sont souvent appelées machines algorithmiques car leur architecture sert à l'interprétation d'un (ou plusieurs) algorithme(s) de base.

Il est classique aussi de considérer un circuit intégré séquentiel (machine séquentielle) comme un ensemble de deux composantes fonctionnelles (comme ceci est démontré dans (18) (47)) :

- partie opérative (PØ),
- partie contrôle (PC).

Chacune des deux composantes assure un rôle spécifique. La réalisation d'un traitement particulier de l'information s'effectue par la coopération des deux composantes :

- la partie opérative réalise le traitement des données,
- la partie contrôle assure le séquencement des actions élémentaires qui se déroulent dans la partie opérative.

Cette décomposition fonctionnelle est tout à la fois naturelle et intuitive elle se retrouve aussi bien au niveau de l' organisme humain (le cerveau étant la partie contrôle, le reste du corps avec ses éléments périphériques tels que les yeux, les mains, .., constituant la partie opérative), que dans la plupart des objets conçus par l'homme, spécialement pour le matériel informatique.

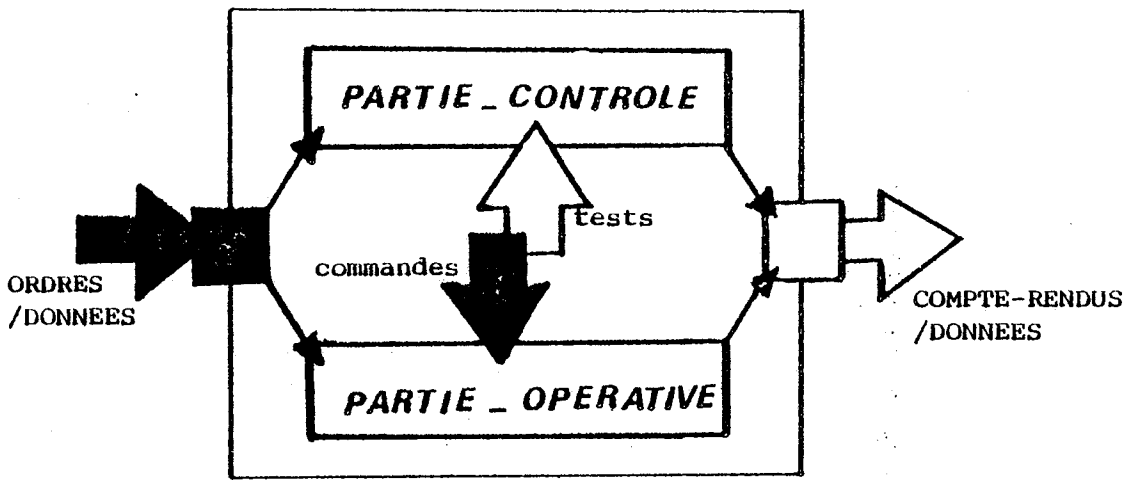


FIG.15_Schema global d'un circuit sequentiel

Notons que cette notion de partie contrôle et de partie opérative n'est qu'une représentation fonctionnelle, car elle car elle n'est pas toujours liée à la topologie reelle des circuits integres.

Cependant, même si cette notion de PØ/PC n'est que fonctionnelle, elle contribue à un assouplissement certain dans la conception de l'architecture globale d'un circuit séquentiel. En effet, la conception matérielle de ces deux composantes peut être réalisée séparément pour chacune d'elles avec des techniques différentes. D'ailleurs, n'applique-t-on pas des methodes differentes pour le developpement de l'organisme humain (culture intellectuelle, culture physique).

2. - Partie contrôle

2.1. - Mécanisme général de contrôle

Le but de ce paragraphe est de présenter un modèle général de machine séquentielle qui nous permet de mettre en évidence les mécanismes généraux des parties contrôle. Pour ce faire, nous choisissons un exemple simple qui permet de clarifier l'exposé tout en mettant à jour les différentes notions (matérielles, fonctionnelles) qui nous paraissent intéressantes.

L'exemple en question est celui d'une station qui est spécialisée dans le traitement des signaux lumineux (décodage, transmission). Des signaux lumineux arrivent à l'entrée de la station, celle-ci se charge de les décoder selon des conventions préétablies. La station procède ensuite à l'émission d'un ensemble de messages lumineux vers l'environnement extérieur selon des règles bien précises (étalement dans le temps, ordre des messages, ...) et ceci en fonction de chaque code (*) des signaux d'entrée.

(*) La combinaison (présence ou absence) des différents signaux lumineux à l'entrée de la station constitue un ensemble de codes

2.1.1. - Notion de séquenceur et d'interface de commande

La station est divisée en deux parties. Une partie est appelée SEQUENCEUR, l'autre partie est appelée INTERFACE DE COMMANDES.

On installe des lampes dans chacune de ces parties selon une certaine organisation (figure 16)

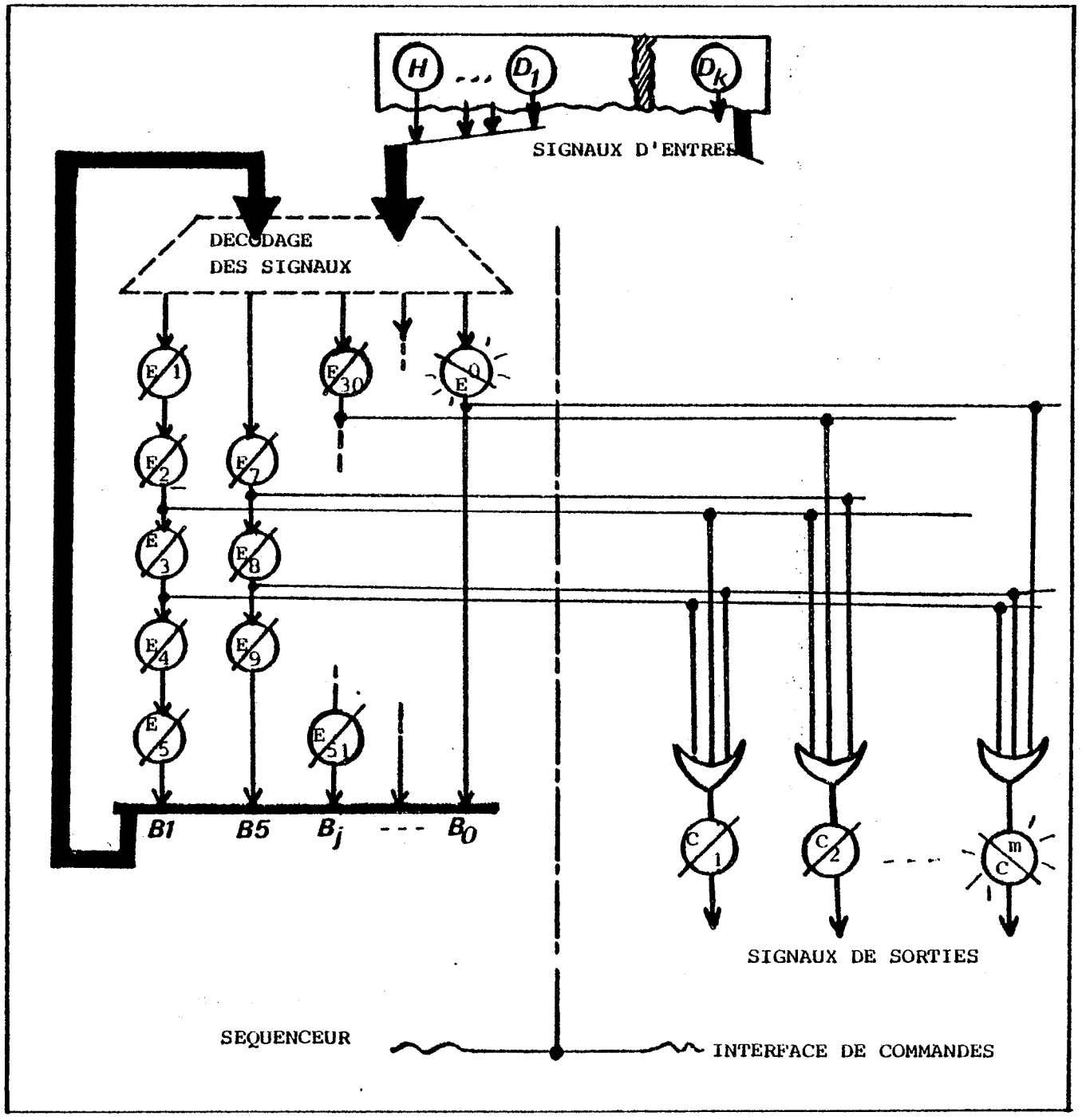


FIG.16_ Structure generale de la station

Chacune des lampes de la partie SEQUENCEUR est identifiée par une étiquette Ei/i = 1, 2, ..., n. De même, chacune des lampes de la partie INTERFACE DE COMMANDE est identifiée par une étiquette Ci/i = 1, ..., m. Les signaux lumineux qui arrivent à l'entrée de la station sont émis par des lampes qui sont identifiées par les étiquettes (H, ..., D1, ..., DK).

La manipulation des lampes (allumage/extinction) dans chacune des deux parties de la station est assurée par un opérateur. L'opérateur de la partie SEQUENCEUR, que l'on désigne par "opérateur S", se charge de la manipulation des lampes (Ei/i = 1, ..., n) en fonction de la lampe courante Ej qui est allumée et du résultat de son observation des signaux lumieux d'entrée. L'opérateur de la partie INTERFACE DE COMMANDE, que l'on désigne par "opérateur I", se charge de la manipulation des lampes (Ci/i = 1, ..., m) uniquement en fonction de son observation des lampes (Ei/i = 1, ..., n) et éventuellement des signaux lumineux d'entrée.

Notons que l'opérateur I possède une caractéristique particulière : il est DALTONIEN (*).

Cette station de traitement des signaux lumineux ainsi définie, correspond à un modèle général de machine séquentielle (*) dans laquelle:

. l'opérateur S qui opère dans la partie séquenceur joue le rôle de la fonction de transition ;

. l'opérateur I, quant à lui, joue le rôle de la fonction de sortie γ ;

. un état interne Qj de la station est déterminé par l'allumage d'une lampe Ej ;

. l'état initial de la station Qo est déterminé par l'allumage de la lampe Eo qui indique le démarrage du fonctionnement de la station ;

Notation:
.Les ronds barrés representent les lampes avec leurs interrupteurs.
.(B1,B2,....,Bj,....,B0) representent des branches de lampes specifiques à chaque traitement.
.Les fleches entre les lampes d'une meme branche determinent l'ordre d'allumage (∫2.12).

—

—

—

. l'ensemble des vecteurs d'entrée est représenté par l'ensemble des signaux lumineux d'entrée (H, ..., D1, ..., DK) ;

. l'ensemble des vecteurs de sortie est représenté par l'ensemble des signaux lumineux de sortie (C1, C2, ..., Cm).

La station est une machine séquentielle du type de Mealy si l'opérateur I manipule les lampes (C1, ..., Cm) en fonction de son observation simultanée des lampes (Ei/i = 1, ..., n) et des signaux d'entrée.

Si l'observation du manipulateur I se limite à celle des lampes (Ei/i = 1, ..., n), alors la station est une machine séquentielle du type de Moore.

La partie SEQUENCEUR de la station (avec son opérateur S) donne des indications précises sur la phase courante du traitement et permet également de déterminer la phase suivante du traitement.

La partie INTERFACE DE COMMANDE (avec son opérateur I) quant à elle ne fait que regrouper les informations et procéder à l'émission de certains signaux selon un protocole prédéfini, sans se préoccuper de ce qui va suivre dans les phases suivantes.

L'essentiel du contrôle étant réalisé dans la partie SEQUENCEUR, l'opérateur I dans la partie INTERFACE DE COMMANDE ne sert que d'intermédiaire pour faciliter la tâche de l'opérateur S.

(*) voir paragraphe suivant.
(*) voir paragraphe VI/11.

2.1.2. Technique de codage des états internes d'une machine séquentielle

Chaque branche du SEQUENCEUR (figure 16) correspond au traitement d'un code des signaux lumineux d'entrée.

A la fin du traitement en cours, l'opérateur S observe certains signaux lumineux à l'entrée de la station et en fonction de leur code, il allume la première lampe (*) de la branche correspondante. Une fois allumée, cette lampe le reste pendant toute la durée d'une phase. La durée d'une phase est déterminée par le temps qui sépare le début de deux apparitions successives d'un signal lumineux d'entrée H (le signal H est généralement appelé horloge).

Le rôle de l'opérateur S (fonctionnement de la partie SEQUENCEUR) se résume comme suit : après l'allumage d'une lampe Ei appartenant à une branche Bj, l'opérateur S ne fait plus rien à part qu'observer le signal H qui doit s'éteindre et s'allumer de nouveau (après la durée d'une phase). Dès la nouvelle apparition du signal H, l'opérateur S procède au changement d'allumage dans la branche correspondante en éteignant la lampe Ei qui était allumée et en allumant la lampe qui la suit dans la même branche. Ceci permet de démarrer une nouvelle phase dans le traitement courant. L'opérateur S opère ainsi jusqu'au bout de la branche correspondante, ensuite il reprend de nouveau la consultation des signaux d'entrée pour allumer la première lampe d'une nouvelle branche, ce qui permet de démarrer un nouveau traitement.

Le rôle de l'opérateur I (fonctionnement de la partie INTERFACE DE COMMANDE) se résume comme suit : à chaque fois que l'opérateur S change allumage des lampes (Ei/i = 1,...,n) dans la partie SEQUENCEUR, l'opérateur I procède à l'extinction des lampes (Ci/i = 1,...,m) qui étaient allumées dans la partie INTERFACE DE COMMANDE et allume de nouvelles lampes et ceci en fonction de la nouvelle lampe qui est allumée dans le SEQUENCEUR.

(**) Cas où la station est bâtie sur le modèle d'une machine séquentielle du type de MOORE.

(*) Pour certains traitements particuliers, plusieurs lampes appartenant à des branches differentes peuvent etre allumées en parallele.

Exemple :

Dans la figure (16)l'allumage de la lampe Eo par l'opérateur S indique à l'opérateur I qu'il faut allumer la lampe Cm et laisser les autres (C1, C2, ..) éteintes. De même, l'allumage de la lampe E3 indique à l'opérateur I qu'il faut allumer les lampes C1 et Cm.

Nous venons d'exposer la structure générale de la station et son fonctionnement ; nous allons maintenant procéder à une optimisation de la quantité du matériel utilisé.

En effet, le coût des lampes (Ei/i = 1, ..., n) et leur installation est très élevé, aussi serait-il intéressant de diminuer leur nombre tout en assurant le même travail.

Notons que le montage de la figure (16)n'est pas optimisé, car on peut diminuer le nombre de lampes pour les branches B1 et B5.

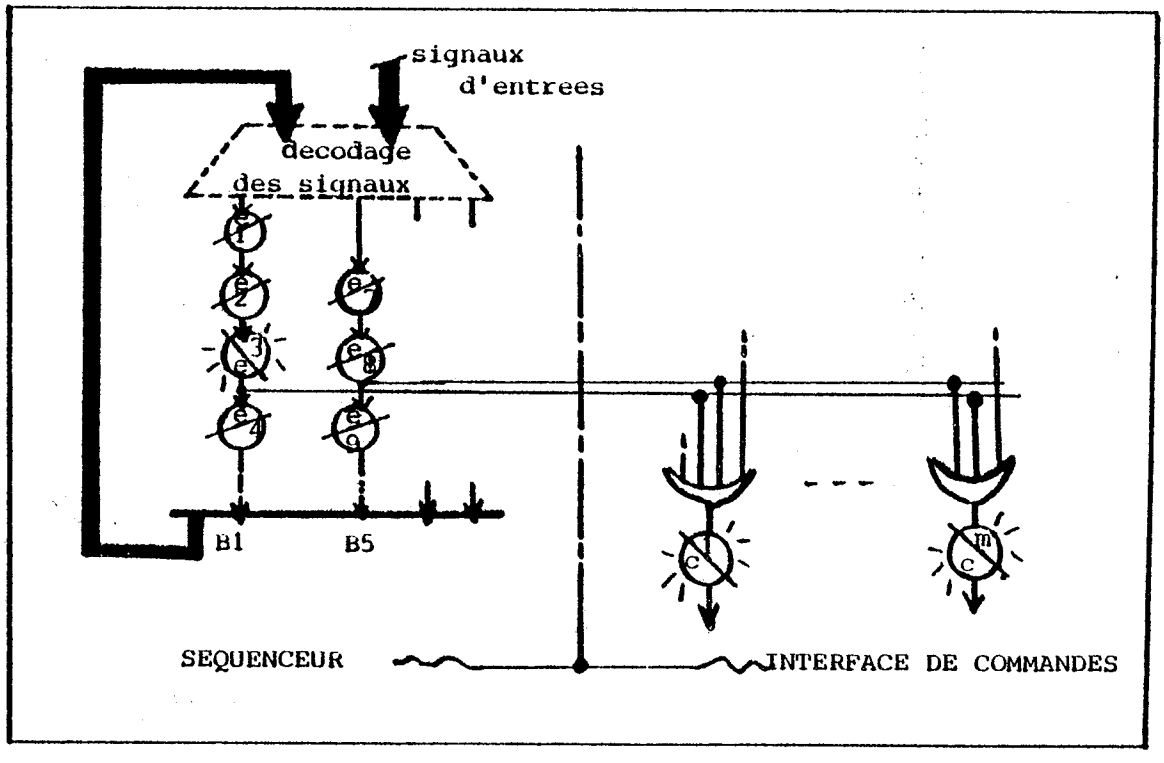


FIG.18 _ Montage des branches B1 ,B5 non optimise (*).

(*) optimisation en nombre de lampes.

La lampe E3 indique la troisième phase du traitement correspondant à la branche B1. La lampe E8 indique la deuxième phase du traitement correspondant à la branche B5. L'allumage d'une de ces deux lampes provoque l'allumage des mêmes lampes dans la partie INTERFACE DE COMMANDE, aussi peut-on les remplacer par une seule lampe que l'on désigne par EB15.

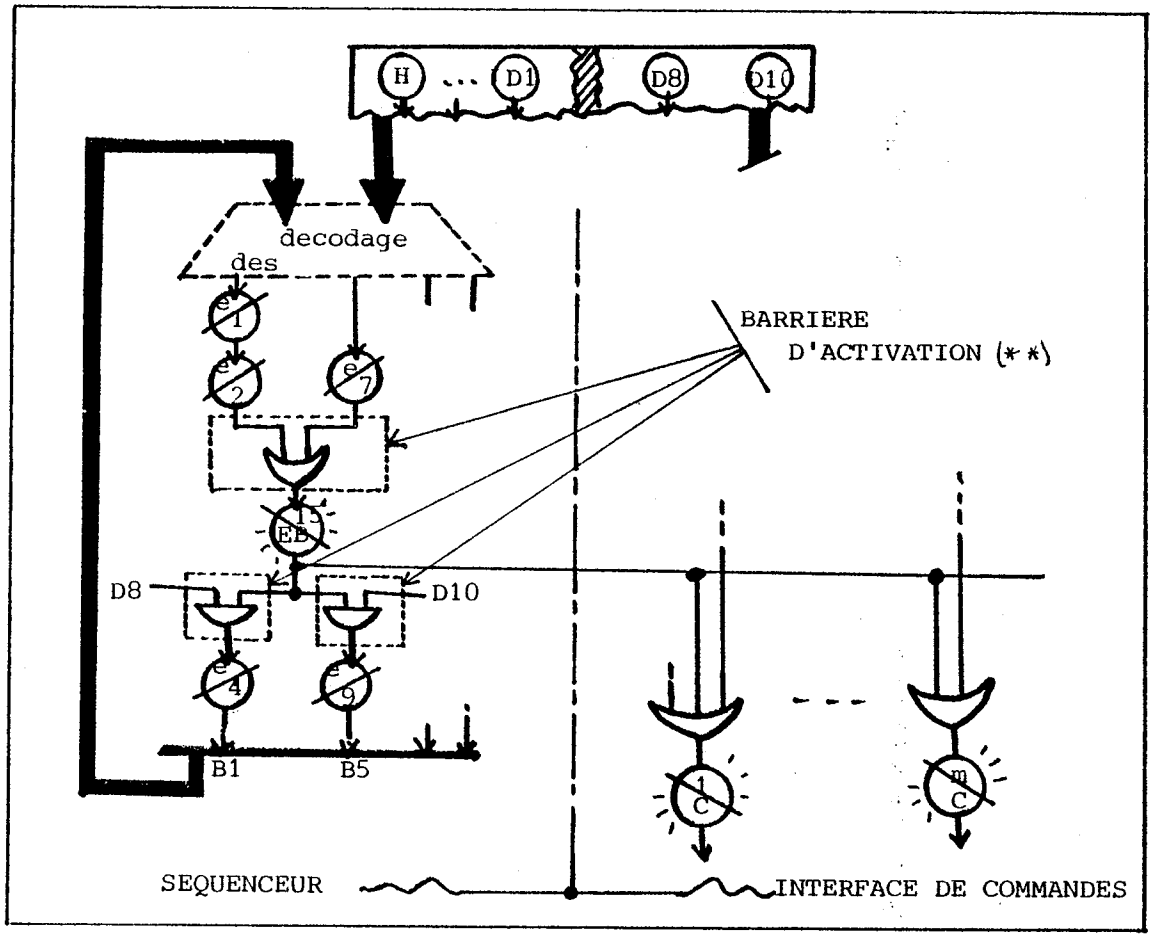


FIG.18- Montage des branches B1, B5 optimisées (*)

(*) Optimisation en nombre de lampes.

(**) Voir définition

La lampe EB15 appartient aux deux branches B1 et B5. Nous introduisons des informations par l'intermédiaire des signaux d'entrée D8 et D10 pour indiquer à l'opérateur S pour le compte de qui la lampe EB15 est allumée pendant les traitements correspondant aux branches B1 et B5. Ainsi dès que la lampe E2 est allumée, ce qui indique la deuxième phase du traitement dans B1 (resp. pour l'allumage de E7 qui indique la premiere phase du traitement dans B5), l'opérateur S procède à l'allumage de la lampe EB15 à l'étape suivante. L'allumage de la lampe EB15 indique que l'on est dans la troisième phase du traitement qui correspond à la branche B1 ou dans la deuxième phase du traitement qui correspond à la branche B5. A la phase qui suit celle de EB15, l'opérateur S doit observer les signaux d'entrée D8 et D10 pour décider s'il faut allumer la lampe E4 ou la lampe E9. Ainsi, si le signal D8 est présent (cela signifie que le traitement en cours est celui qui correspond à la branche B1), alors l'opérateur S procède à l'extinction de EB15 et allume la lampe E4. Si par contre c'est le signal D10 qui est présent, alors l'opérateur S éteint la lampe EB15 et allume la lampe E9 ce qui indique que l'on est dans la troisième phase du traitement correspondant à la branche B5.

Pour l'opérateur I, dans la partie INTERFACE DE COMMANDE, rien n'est changé. Chaque fois que la lampe EB15 s'allume, il procède à l'allumage des lampes C1 et Cm sans se demander (*) de quel traitement il s'agit (celui de la branche B1 ou celui de la branche B5).

L'opération d'optimisation qui a été réalisée sur les branches B1 et B5 peut être étendue à l'ensemble des branches de la partie SEQUENCEUR et ainsi le nombre total des lampes va en diminuant avec l'apparition d'un certain nombre de barrières d'activation. Le résultat final donne un montage moins coûteux (encombrement, temps de réalisation).

(*) On est toujours dans le cas d'une machine séquentielle du type de MOORE.

Si maintenant on décide d'utiliser des lampes de couleurs différentes dans la partie SEQUENCEUR, cela permet de donner des indications plus générales sur le traitement en cours.

Exemple :

On choisit des ampoules de couleur verte pour toutes les lampes qui indiquent la première moitié des phases d'un traitement et des ampoules de couleur rouge pour les lampes des autres moitiés. Ainsi, dans une branche du séquenceur qui contient n lampes, les $\lfloor n/2 \rfloor$ (*) premières lampes émettent une lumière verte quand elles sont allumées et le reste des lampes de cette même branche émettent une lumière rouge. Ainsi un observateur M qui serait à l'extérieur de la station et ne s'intéressant qu'à la couleur de la lampe qui est allumée dans la partie séquenceur, saura dans quelle étape est le traitement courant :

. l'opérateur I est en train d'émettre la première moitié du message correspondant au traitement courant si la lumière est verte ;
. l'opérateur I est en train d'émettre la deuxième moitié du message correspondant au traitement courant si la lumière est rouge.

On se doute bien que toutes ces nouvelles indications qui sont introduites au niveau du séquenceur, laissent indifférent l'opérateur I dans la partie INTERFACE DE COMMANDE, car comme on l'a précisé dans le paragraphe précédent, celui-ci est DALTONIEN. Par contre, pour ce qui est du problème d'optimisation du nombre de lampes dans les branches du SEQUENCEUR, on décide d'effectuer cette opération uniquement entre les lampes qui possèdent des ampoules de la même couleur.

Tout ceci met en évidence une notion de decomposition d'interpretation concernant un traitement.

- 1er TYPE d'interprétation :
  l'observateur M s'intéresse uniquement à l'étape courante d'un traitement sans se préoccuper de quelle phase de l'étape il s'agit et ceci pour toutes les branches du SEQUENCEUR.

(*) $\lfloor n/2 \rfloor$ représente la partie entière de n/2.

- 2ème TYPE d'interprétation :
  que le traitement en cours soit dans sa première étape ou dans la deuxième, ce qui intéresse l'opérateur S est l'ordre de la phase courante pour déterminer la phase suivante et ceci pour chaque branche du SEQUENCEUR.

- 3ème TYPE d'interprétation :
  l'opérateur I quant à lui, s'intéresse uniquement à l'identification de la phase courante pour émettre les signaux (Ci/i = 1, ..., m) correspondants, sans se demander si celle-ci appartient à la première étape ou à la deuxième, ou encore qu'elle soit la troisième ou la quatrième phase de tel ou tel traitement.

L'introduction de nouvelles caractéristiques dans le fonctionnement de la station permet de définir de nouveaux TYPES d'interprétation.

2.2. - Aspect matériel des parties contrôle

Nous présentons les différentes structures des parties contrôle pour matérialiser les notions mises en évidence dans le paragraphe précédent.

Notons que ces différentes notions sont communes à l'ensemble des parties contrôle, indépendamment de l'organisation matérielle de telle ou telle réalisation.

Il existe toute une variété de structures des parties contrôle dont le choix au niveau de la réalisation dépend de divers critères tels que le coût de la conception, la facilité de leur implantation sur du silicium, la densité d'intégration ...

Deux types de structure sont exploités pour la réalisqtion des différentes variantes de parties contrôle, ce sont :

- les structures câblées (ou structures anarchiques),
- les structures à algorithme enregistré (ou structures régulières).

Nous désignons par structure à algorithme enregistré, aussi bien les structures microprogrammées que les structures à base de PLA.

La plupart des réalisations actuelles sont effectuées à partir de structures mixtes, c'est-à-dire une combinaison de ces deux types de structures.

2.2.1. Structures cablées :

Ce type de structures présente beaucoup d'avantages par sa grande densité d'implantation.

Cette technique de cablage a été abondamment utilisée par le passé. Cependant, son usage devient de plus en plus spécialisé et s'il continue de l'être, c'est simplement à cause des contraintes liées à la taille des circuits : cas du Z8000 (*) par exemple.

Ceci est dû essentiellement au coût élevé que demande leur implantation. En effet, l'implantation de ces structures anarchiques est réalisée en grande partie manuellement. Comme la taille des circuits (en nombre de transistor) ne cesse de croître, seule une solution spectaculaire apportée au problème de l'implantation automatique peut redonner un intérêt de premier plan à ce type de structures. Ceci n'est pas impossible, une démarche qui serait basée sur les techniques d'implantation manuelle utilisées par les concepteurs peut aboutir à des outils efficaces pour l'implantation automatique de ces structures cablées, une approche intéressante de ce problème est présentée dans (4) (16).

a) caractéristiques particulières des structures cablées :
   Le mécanisme général du fonctionnement des parties contrôles à structures cablées a été présenté à travers l'exemple précédent de la station qui traite des signaux lumineux à part que pour les parties contrôles, il s'agit de traiter des signaux électriques.

---

* Microprocesseur 16 bits réalisé par ZILOG, sa surface totale est d'environ 38.9 mm (≃ 17500 transistors) (36)

Nous allons maintenant présenter les différents aspects matériels qui caractérisent la partie contrôle à structure cablée d'un circuit séquentiel synchrone. Pour cela, nous nous servons toujours de l'exemple de la station en remplacant les objets de celle-ci par ceux utilisés dans la conception d'une partie contrôle à structure cablée et ceci dans chaqu'une des deux parties.

AU NIVEAU DU SEQUENCEUR :

Les différentes branches de lampes sont remplacées par des registres à décalage.
Chaque étape d'un registre à décalage est constitué par une bascule (Maître/Esclave). Un état interne de la machine qui était représenté par l'allumage d'une lampe Ei sera représentée, cette fois, par la mémorisation d'une charge électrique dans la partie maître (Ei) de l'étape d'un registre à décalage. Ainsi, chaque branche (ou portion d'une branche, fig.18) sera représentée par un registre à décalage. Aussi, la propagation (*) de l'information à travers les étapes d'un registre à décalage peut être conditionnée par certains paramètres provenant des signaux électriques d'entrées.

(*) cette propagation est commandée par l'horloge du circuit séquentiel et traduit le changement de l'état interne de la machine.

Exemple :

Nous représentons la portion de la partie contrôle qui correspond au montage du séquenceur.

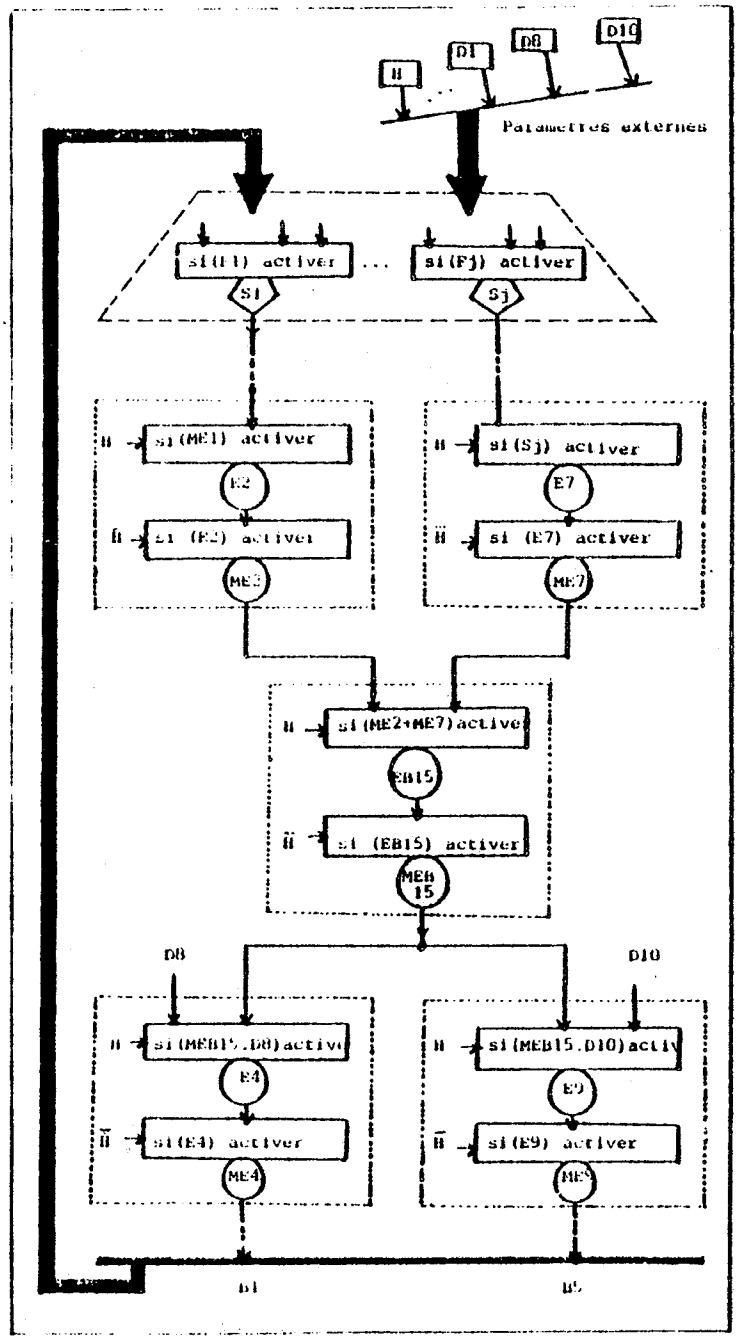


FIG.19A_Description DELTA

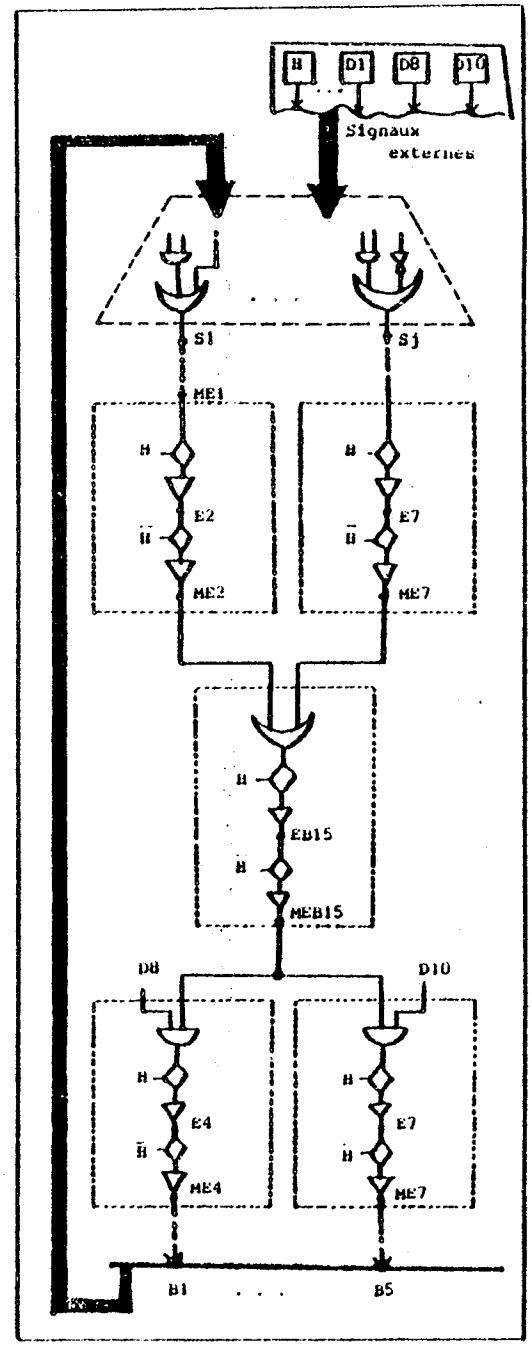


FIG.19B_Description logique
(Elements de mémorisation dynamique)

AU NIVEAU DE L'INTERFACE DE COMMANDE :

Les signaux lumineux de sortie dans l'exemple de la station sont remplacés par des signaux électriques que l'on désigne par : commande.

Les commandes sont générées essentiellement par des circuits combinatoires. Cependant pour des réalisations particulières, on peut vouloir mémoriser les commandes et ainsi pouvoir retarder leur envoi à la partie opérative.

Ceci est utilisé surtout dans le cas des machines séquentielles PIPE-LINE* Ces caractéristiques sont valables pour toutes les machines séquentielles quelque soit leur type (MEALY, MØØRE).

Notons que selon le type des éléments de mémorisation, une partie contrôle est dite :

- dynamique si elle possède au moins un élément de mémorisation dynamique (*)
- statique si tous ses éléments de mémorisation sont statiques (*)

La notion des différents niveau de decomposition existe au niveau fonctionnel des parties contrôles à structure cablée. Par exemple, pour les circuits séquentiels du type microprocesseur, cette notion se traduit dans l'exécution d'une instruction, par la décomposition en plusieurs étapes de traitement (fig. 20).

(*) voir définition et exemple dans la PARTIE.4

- Etape de chargement de l'instruction (ou séquence acquisition),
- Etape de chargement des opérandes (ou séquence d'adressage),
- Etape de calcul (ou séquence opération).

Cependant, cette notion des différentes décompositions d'interprétation n'est pas du tout apparente dans la topologie des circuits séquentiels à structure cablée, car leur implantation se fait d'une manière tout à fait anarchique.
Il en est de même pour la séparation entre les parties SEQUENCEUR et INTERFACE DE COMMANDES.

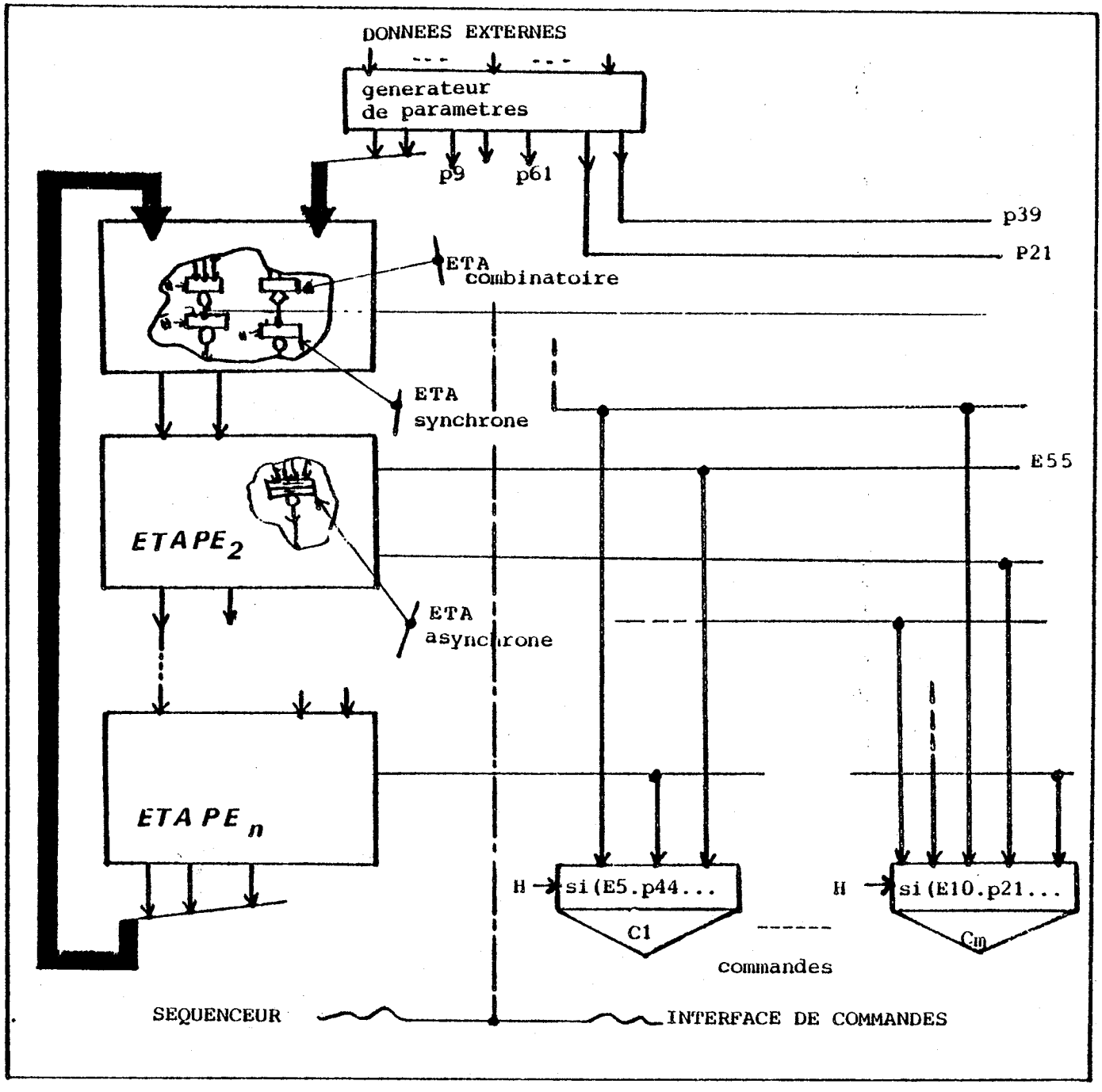


FIG.20_Description DELTA de l'architecture générale d'une partie controle à structure cablée (model de Mealy).

a)- Notion de logique positive et logique négative

Soient deux registres à décalage (R1, R2) dont les etages sont conçu suivant une structure de (maître/esclave). La propagation de l'information au niveau de ces deux registres est assurée par une horloge biphase (Ø1, Ø2).

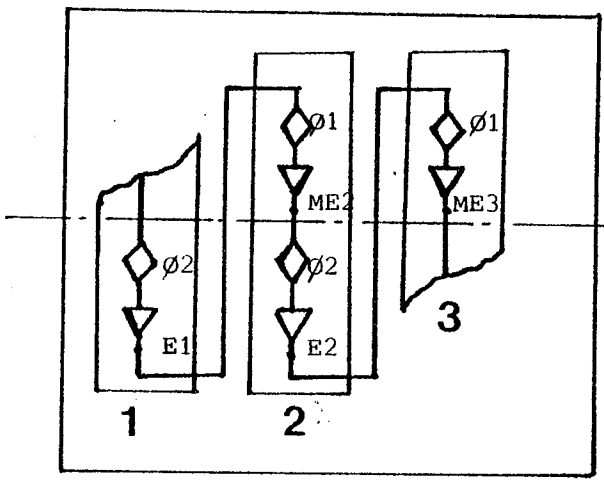


FIG. 21 A - Registre à décalage R1.

FIG. 21 B - Registre à décalage R2.

Les parties(maître)et(esclave) du registre à décalage R1 sont réalisées à base de points mémoire dynamique qui n'inversent pas les valeurs logiques des entrées (deux couches d'inverseurs).

Les parties(maître)et(esclave)du registre à décalage R2 sont réalisées à base de points mémoire dynamique qui inversent les valeurs logiques des entrées (une seule couche d'inverseurs, FIG.21B)

Ainsi, si on injecte la valeur logique '1' (resp. '0') à l'entrée d'un étage du registre R2, la sortie de celui-ci prend la valeur logique '1' (resp. '0') au cycle suivant.

Ceci se réalise par la propagation de l'information dans l'étage correspondant après une double inversion logique.

La première inversion s'effectue dans la partie maître (un inverseur), la deuxième inversion s'effectue dans la partie esclave (un inverseur).

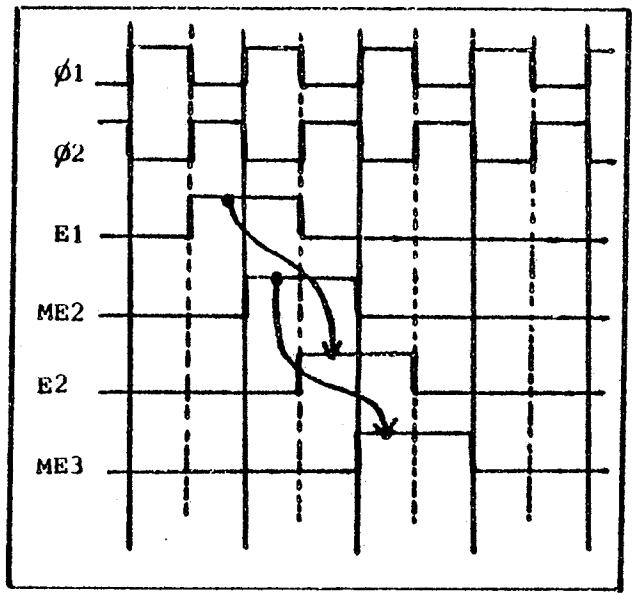


FIG. 22 A - Diagramme des temps de R1

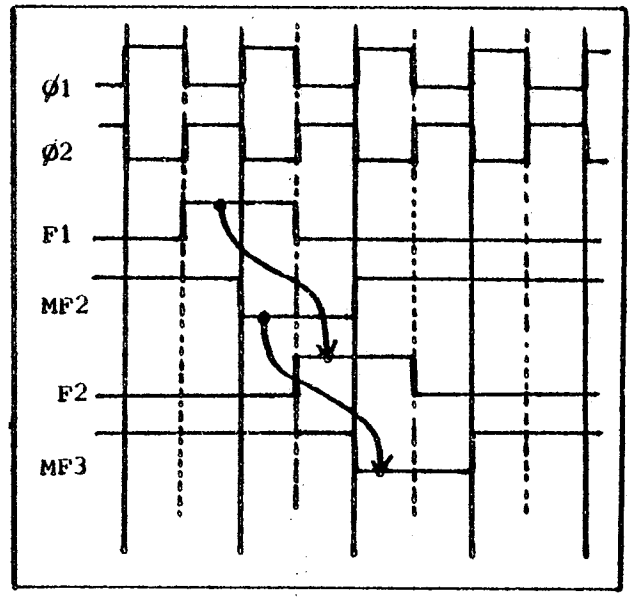


FIG. 22 B - Diagramme des temps de R2

Sur les diagrammes de temps Figures 22 (A, B), on observe une simillitude entre l'action du signal E1 sur E3 dans le registre R1 et celle du signal F1 sur F3 dans le registre R2.

Il en est de même pour l'action des signaux (ME1, MF1) sur les signaux (ME3, MF3).

En effet, dans le registre R1, le passage à '1' du signal E1 en Ø2 du cycle Tn, permet d'abord le passage à '1' du signal ME2 en Ø1 du cycle Tn+1 (sans inversion logique), ensuite le passage à '1' du signal E2 en Ø2 du cycle Tn+1 (également sans inversion logique).

De même, dans le registre R2, le passage à '1' du signal F1 en Ø2 du cycle Tn, permet le passage à '0' du signal MF2 en Ø1 du cycle Tn+1 (première inversion logique) ensuite le passage à '1' du signal F2 en Ø2 du cycle Tn+1 (deuxième inversion logique).

L'inversion des valeurs logiques lors du transfert des informations entre les deux parties (maître/esclave) d'un étage du registre R2, détermine deux types de logique :

- une logique positive dans les parties esclaves du registre R2,
- une logique négative dans les parties maîtres du registre R2.

Le registre R1 fonctionne en logique positive au niveau des deux parties (maître/esclave).

Dans une logique positive, la propagation de l'information "utile" se fait par la valeur logique '1'. Dans une telle logique, un signal (E) est dit actif (*) si on a: (E=1).

Dans une logique négative, la propagation de l'information "utile" se fait par la valeur logique '0'. Dans une telle logique, un signal (E) est dit actif si on a: (E=0).

(*) Signal qui déclenche une action.

Le passage d'une logique positive à une logique négative (et réciproquement) se fait d'après les règles de DE MORGAN.

L'introduction de cette notion de logique positive et logique négative dans la réalisation d'une machine séquentielle, permet un gain appréciable en surface d'implantation.

Par exemple, la partie controle du microprocesseur 6800 est entièrement dynamique. La notion de logique positive et de logique négative y est exploitée aussi bien au niveau du séquenceur que dans l'interface de commandes. Les conventions adoptées sont :

pour le séquenceur :

- une logique positive en Ø1,
- une logique négative en Ø2.

pour l' interface de commandes :

- une logique négative en Ø1,
- une logique positive en Ø2.

2.2.2. Structures à Algorithme enregistré :

Les caractéristiques fonctionnelles et matérielles des structures microprogrammées et à PLA sont très semblables.

La principale différence réside au niveau du degré d'optimisation des différentes couches combinatoires, donc de la surface totale occupée par la partie contrôle. En effet, pour une même partie contrôle, la surface occupée par une structure microprogramme est plus importante que celle d'une structure à PLA. Cependant l'avantage connu à ces deux structures, par rapport aux structures cablées, est la grande facilité de leur implantation. Ceci est dû à la grande régularité dans l'organisation des différentes couches combinatoires (fig. 23).

De la même façon que pour les parties contrôles à structure cablée, nous allons maintenant présenter les caractéristiques matérielles des structures de parties contrôles à Algorithme enregistré.

AU NIVEAU DU SEQUENCEUR :

Les lampes, dans le cas de la station, sont remplacées par des éléments de mémorisation, pour les structures microprogrammées, chaque lampe (Ei) est remplacée par un élément de mémorisation (Mi) (Fig.23). Pour les structures à base de PLA, les lampes Ei, qui sont numérotées de 1 à N, sont représentées par des éléments de mémorisation (Mi) (Fig.23). Ces éléments de mémorisation indiquent le numéro (en Binaire) de chaqu'une des N lampes. Aussi, le nombre des éléments de mémorisation Mi qui serait nécessaire est log.2 (2. [N/2+1])*

---

* [N/2+1] représente la partie entière de (N/2 + 1)

Chaque état interne de la partie contrôle sera défini par la mémorisation d'une (ou plusieurs) charge électrique dans les éléments de mémorisation (Mi/i = 1,...,n) (fig. 23) :

- Activation d'un mot mémoire* pour les structures à base de PLA.
- Activation d'un seul élément de mémorisation (Mi) (fig. 25) pour les structures microprogrammées.

La notion d'état interne de la partie contrôle est souvent associée à celle de Micro-instruction. Les éléments de mémorisation (TMi/i = 1, ...k) (fig.23) qui constituent un registre, assurent le rôle d'une barrière temporelle. Ce registre est souvent appelé : registre d'adresse de la micro-instruction suivante.

AU NIVEAU DE L'INTERFACE DE COMMANDE :

La génération des commandes se fait essentiellement à travers des couches de circuits combinatoires (ET, ØU). Cependant, comme pour les structures cablées, certaines réalisations nécessitent la temporisation des commandes (retardement) dans l'interface de commande avant leur envoi à la partie opérative et l'environnement extérieur. Aussi, les éléments de mémorisation qui assurent ce rôle constituent un registre qui est souvent appelé : registre de la micro-instruction courante.

---

* Mise à "1" de plusieurs éléments de mémorisation (Mi) (Fig. 26)

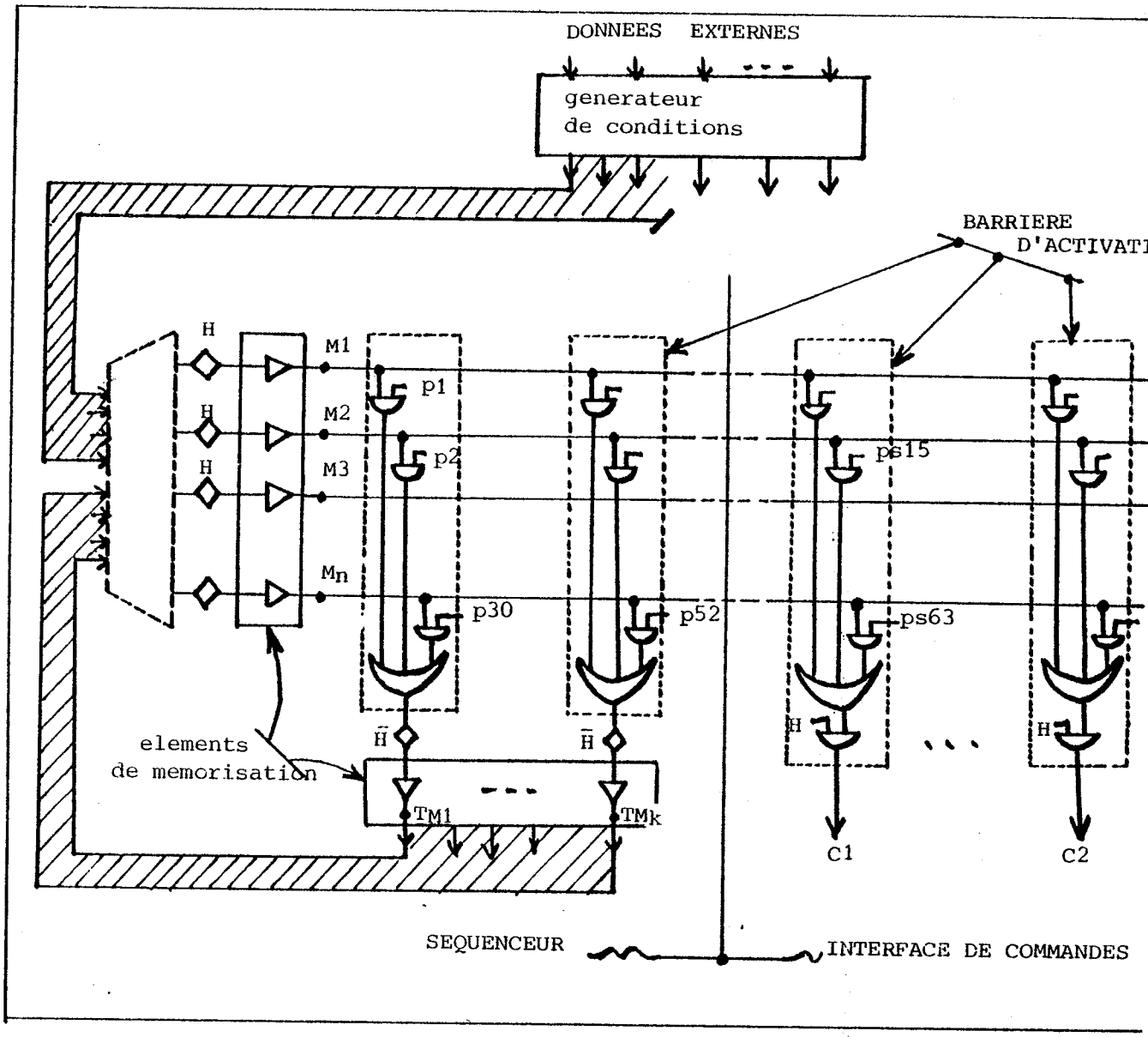


Fig. 23 : Description logique des parties contrôles, à Algorithme enregistr

La matérialisation de la notion des différentes décompositions dans l'interprétation des données externes peut être réalisée par l'empilement* dans le séquenceur d'un certain nombre de blocs semblables à celui du montage du Séquenceur de la figure (23).

La séparation entre le Séquenceur et l'interface de commande est bien apparente au niveau de la topologie des parties contrôles à algorithmes enregistrés.

---

* Les sorties TMI... d'un bloc (i) servent d'entrée pour un bloc (i + 1)

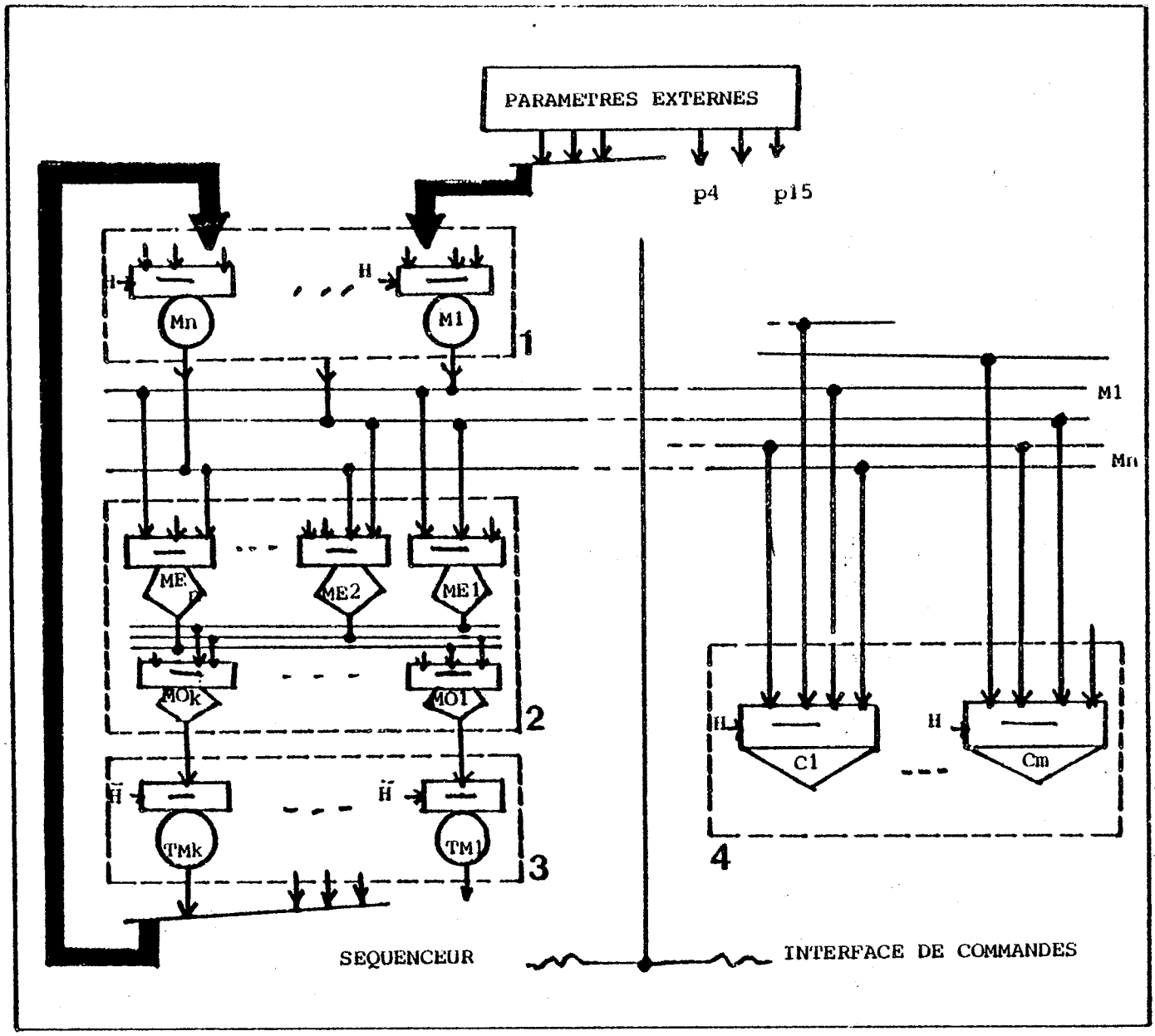


Fig. 24 : Description DELTA globale des parties contrôles à Algorithme enregistré

La description DELTA de la figure (24) a été réalisée d'une manière détaillée au niveau du SEQUENCEUR pour permettre l'accès direct à tout point de celui-ci, notamment pour la partie 2 (fig.24) qui correspond à la description d'une RØM ou d'un PLA (avec ses deux couches combinatoires ET, ØU).

Cependant, une description DELTA moins détaillée peut être générée au Niveau du Séquenceur sans problème comme cela est fait pour la description de l'INTERFACE DE COMMANDE, pour une question de clarté, les expressions booléennes des barrières d'activation de la description DELTA (fig.24), n'ont pas été précisées.

Nous allons maintenant donner la forme générale de ces différentes expressions booléennes :

- la partie (1) correspond à la description DELTA du registre contenant l'adresse de la micro-instruction courante. Chaque bit Mi de ce registre est décrit comme suit :

<H> Mi ← Ei (Pj, ..., TM_k) ;

* Ei étant une fonction booléenne*

- la partie (2) correspond à la description DELTA soit des Monômes d'une RØM (cas des structures microprogrammées) soit à celles des monômes d'un PLA (cas des structures à base de PLA) et ceci pour chacune des deux couches (ET, ØU) *

La description DELTA d'un monôme MEi de la couche (ET) est :

$$MEi = \prod_{j=1}^{n} (K1_j + M_j) ;$$

La description DELTA d'un monôme MOi de la couche (CØU) est :

$$MOi = \sum_{j=1}^{z} (K2_j - ME_j) ;$$

- la partie (3) représente la description DELTA du registre d'adresse de la micro-instruction suivante.
La description DELTA d'un bit de ce registre est :

<¬H> TMi ← MOi ;

---

* voir les paragraphes (a) (b) qui suivent

- la partie (4) représente la description DELTA du PLA ou de la RØM (selon la structure choisie) dont les sorties sont les commandes de la partie contrôle.

La forme générale de la description DELTA d'une commande Ci est :

$$\langle H \rangle \quad Ci := \sum_{k=1}^{mo} \left[\left(\prod_{j=1}^{n} (\alpha 1\, k_j + M_j) \right) . \prod_{j=1}^{no} (\alpha\, 2k_j + P_j) \right) . B_R \right]$$

Notons que la description DELTA des commandes Ci est réalisée d'une manière condensée. L'éclatement de la barrière d'activation peut être réalisée facilement comme cela a été fait pour la RØM (ou le PLA) dans le SEQUENCEUR

Note :

Les paramètres (K1j, K2j, α1Kj, α2Kj, βk) permettent à l'utilisateur de programmer d'une manière très facile les PLA RØM du SEQUENCEUR et de l'INTERFACE DE COMMANDE.

a) Caractéristiques particulières des structures micro-programmées :

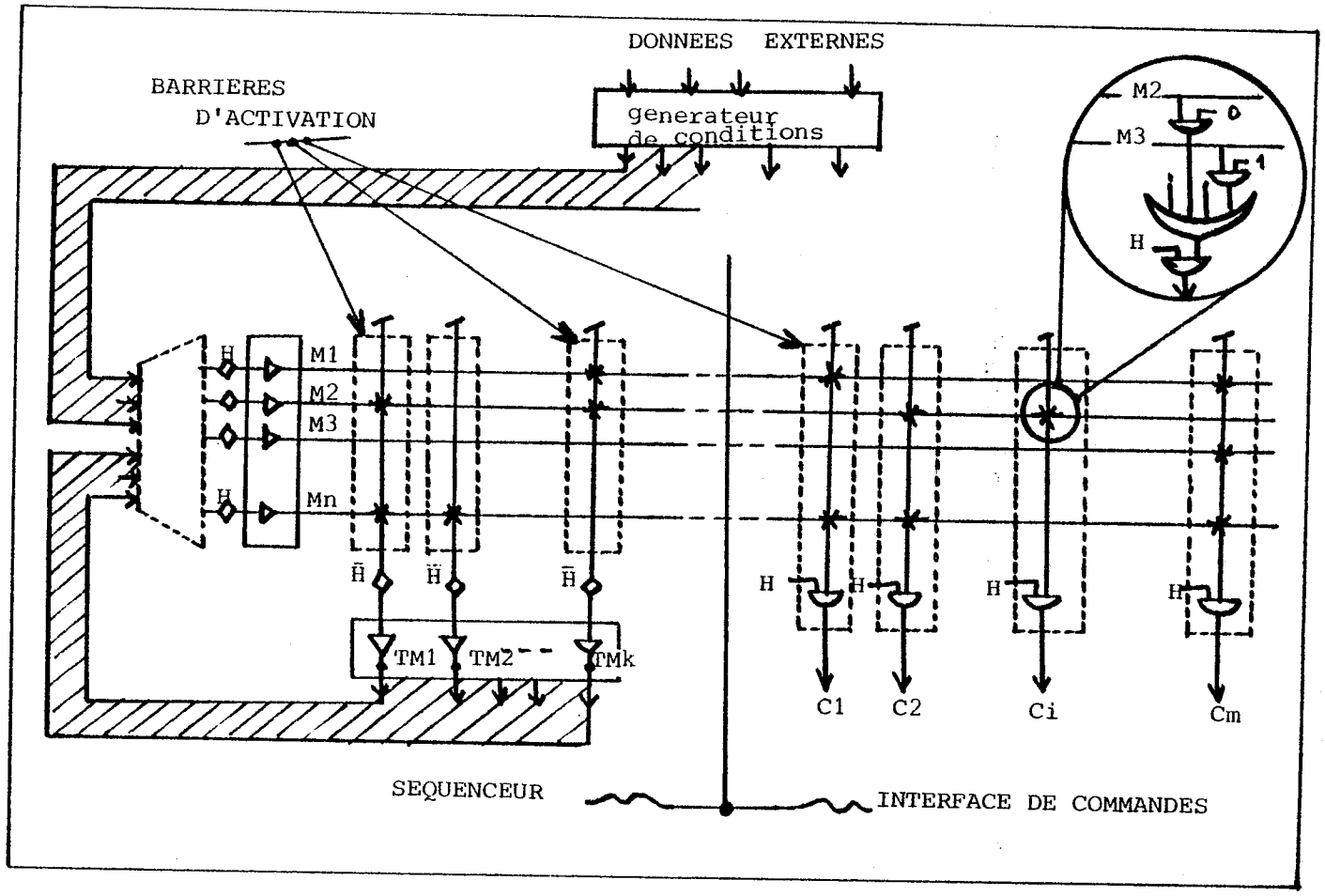


fig. 25 : structure générale d'une partie contrôle micro-programmée

Ce schéma est un cas particulier de celui de la figure (23) dont les constantes (pi) ont été positionnées à '1' ou à '0' selon les besoins.

Le passage d'un état interne Ei à un état interne Ej pour les structures micro-programmées s'effectue comme suit :
soit Mi un élément de mémorisation activé pendant la première phase de l'horloge(H), alors pendant la deuxième phase ($\overline{H}$), l'adresse de la micro-instruction suivante est déterminée par des barrières d'activation du SEQUENCEUR et mémorisée dans le registre d'adresse TM. Ensuite, pendant

la première phase suivante (H), la couche combinatoire (Bo) permet d'activer un élément de mémorisation Mj (et un seul) et ceci à partir du décodage des signaux électriques issus du registre d'adresse TM et de ceux d'un ensemble de conditions liées aux données externes.

b) Caractéristiques particulières des structures à base de PLA :

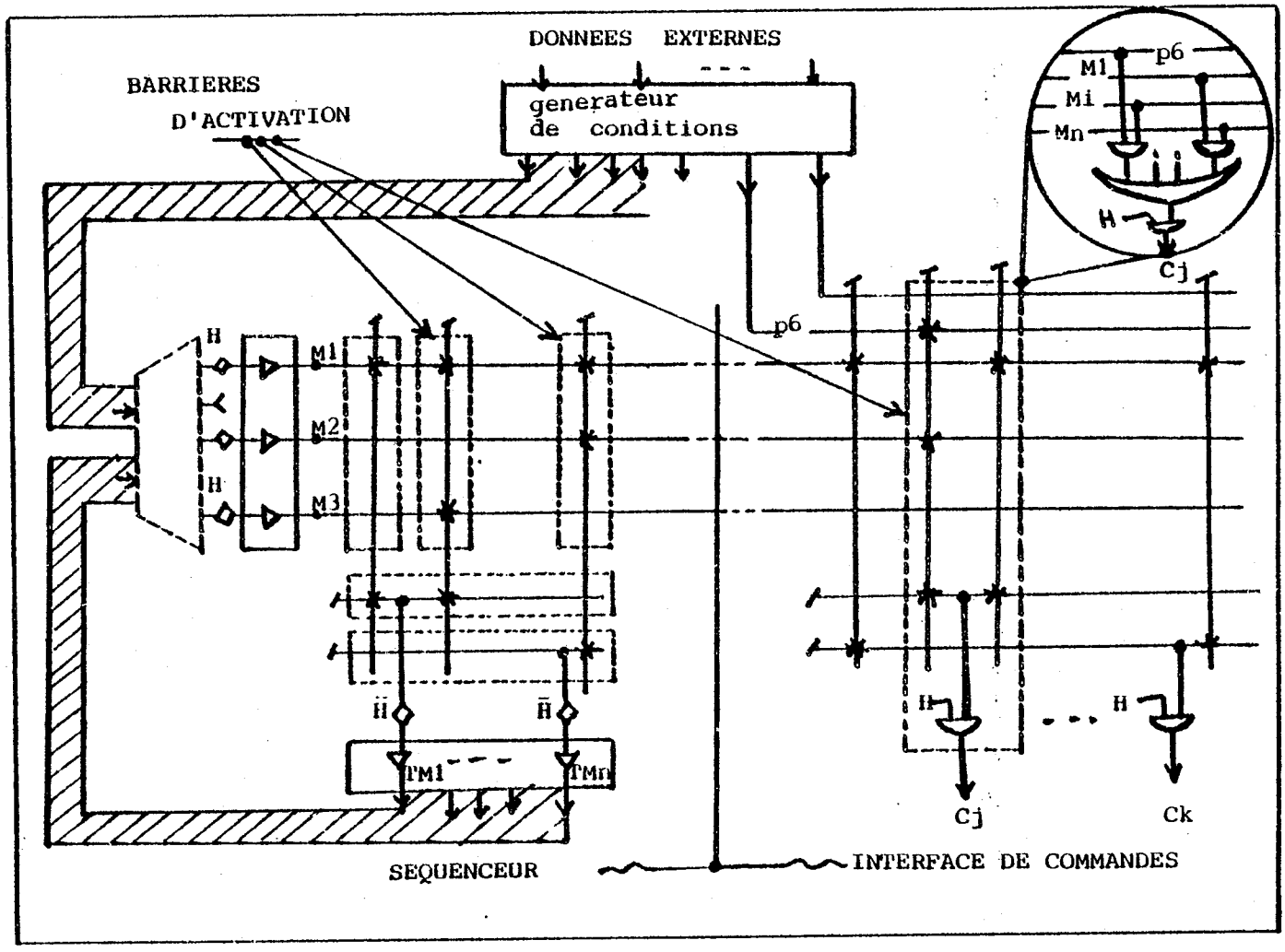


Fig.26 : structure générale d'une partie contrôle à base de PLA

Ce schéma est également un cas particulier de celui de la figure (23) dont les paramètres (Pi) sont issues du générateur de conditions qui peut être constitué par un PLA. Le nombre d'éléments de Mémorisation (Mi) est égal à celui des éléments de Mémorisation (TMi).

Le passage d'un état interne à un autre état interne, pour les structures à base de PLA, est semblable à celui des structures microprogrammées à la seule différence qu'un état interne, pour les structures à PLA, est représenté par l'activation de plusieurs éléments de mémorisation (Mk) alors que pour les structures microprogrammées, un état interne est représenté par l'activation d'un seul élément de mémorisation (Mk).

3.- PARTIE OPERATIVE D'UN CIRCUIT SEQUENTIEL

3.1. Notion d'unité de traitement

Une unité de traitement est définie par le regroupement d'un ensemble d'opérateurs et de variables (registre, additionneur, bus,...) qui contribuent à la réalisation d'un traitement donné.

Le nombre, le type et l'organisation de ces différents opérateurs, au sein d'une même unité de traitement, sont choisis de façon à répondre efficacement aux besoins de tel ou tel traitement des données.

Il existe plusieurs modules d'architecture pour la conception d'une unité de traitement. Les caractéristiques de ces modèles sont fonction du nombre de bus et de leur largeur , du type de l'opérateur UAL (Unaire, binaire)...

Une partie opérative possède une ou plusieurs unités de traitement.

Les différentes unités de traitement d'une partie opérative peuvent-être conçues pour travailler en parallèle et donc permettre d'améliorer les performances du traitement des données. Ainsi, par exemple, dans un micro-processeur la partie opérative est composée essentiellement de deux types d'unités de traitement :

- Unité(s) de traitement des données
  (opération arithmétique et logique,...),
- Unité(s) de traitement des adresses
  (opération de branchement, de saut,...).

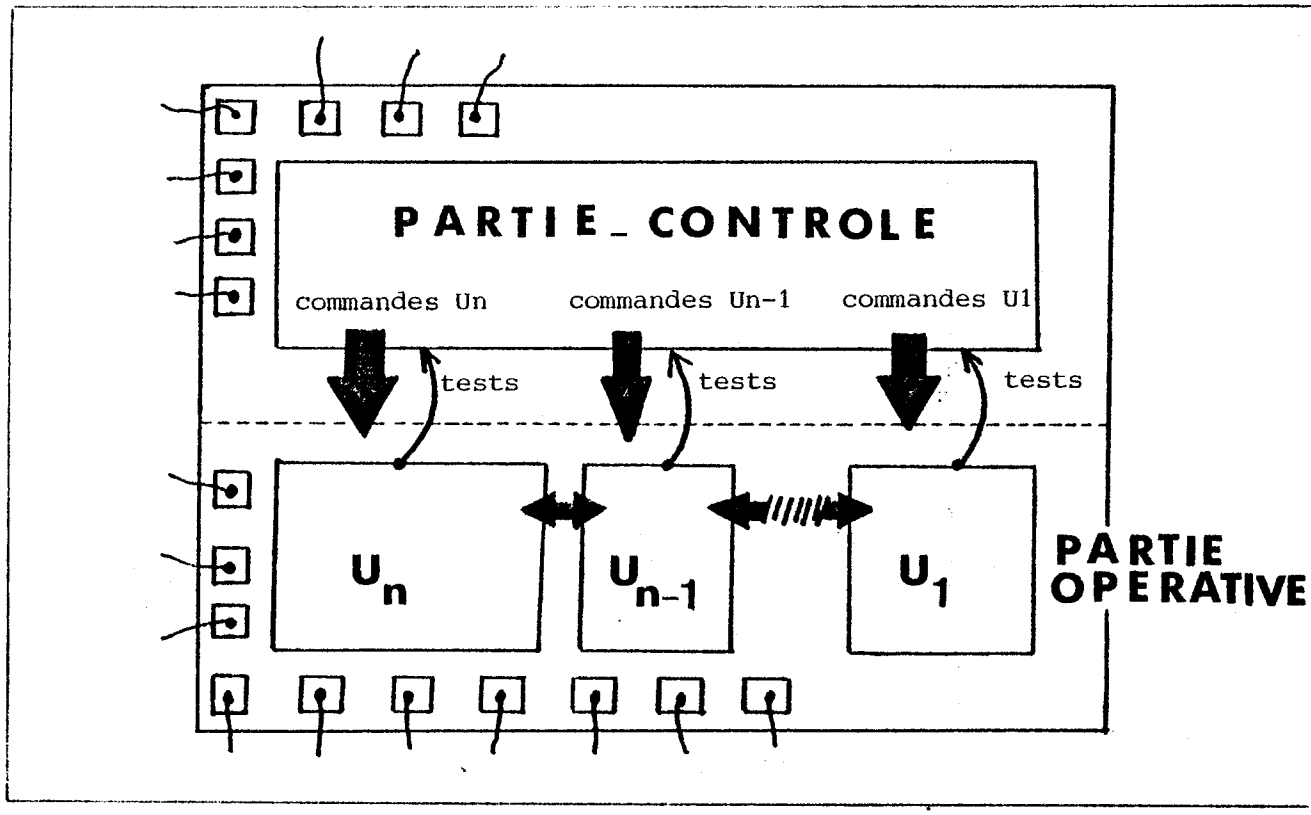


Fig. 27 : Décomposition d'une partie opérative en Unité de traitement.

Cette technique de décomposition, de la partie opérative, en un ensemble d'unité de traitement, est très intéressante, notamment pour faciliter et simplifier les problèmes liés à la conception de l'architecture générale du circuit séquentiel.

- Structure régulière dans la partie opérative,
- Localisation et une meilleure organisation des commandes au niveau de l'interface de commandes de la partie contrôle.

Cependant, pour certains circuits, cette notion d'unité de traitement, bien qu'elle soit exploitée au niveau de l'architecture globale, n'est pas apparente du point de vue topologique.

La tendance actuelle et celle de l'exploitation intensive de cette notion dans toutes les étapes de la conception (Architecturale et topologique).

Par exemple, dans la gamme des microprocesseurs 16 bits, le dernier né qui est le MC 68 000 de MOTOROLA possède trois unités de traitement (15) :

- deux unités pour le traitement des adresses,
- une unité pour le traitement des données (opérandes)

3.2. Mécanisme Général du Traitement des Données

La réalisation du traitement des données au niveau de la partie opérative s'effectue par la coopération.
d'une part, de l'ensemble des opérateurs d'une unité de traitement ;
d'autre part, par un échange éventuel des informations entre les différentes unités de traitement.

Vu du côté de la partie Contrôle, les traitements dans une partie opérative se résume à un ensemble d'aiguillages, des données d'un ensemble d'opérateurs sources vers un ensemble d'opérateurs destinataires.

Cet ensemble d'aiguillages est réalisé à travers des couches combinatoires. Ces différentes couches combinatoires constituent des barrières dont l'ouverture ou la fermeture est conditionnée par les commandes issues de la partie contrôle.

Nous allons visualiser cette notion de transfert des données par un exemple de traitement dans une unité de traitement. Pour cela, nous présentons une structure générale d'une unité de traitement fig. 28.

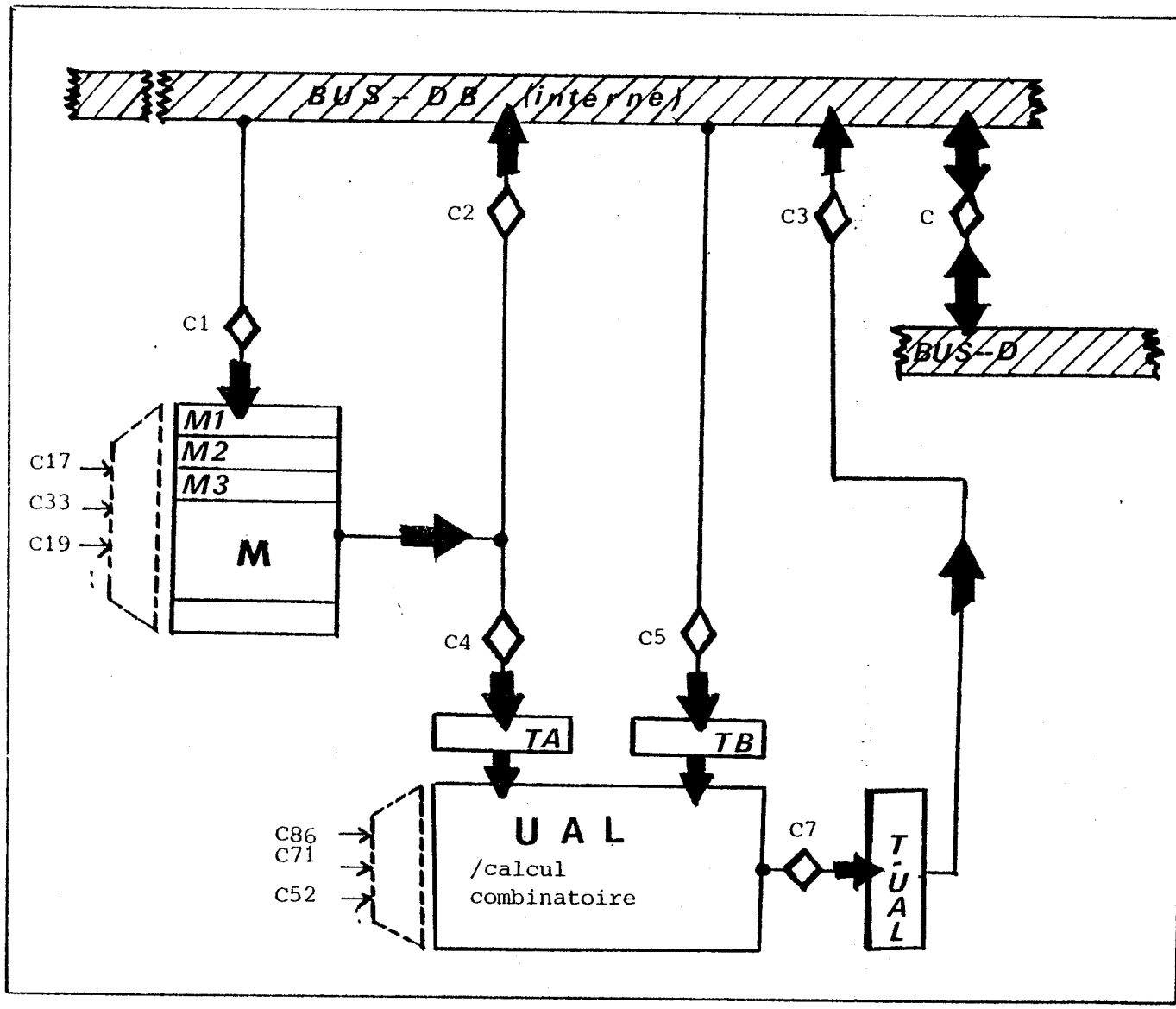


Fig. 28 : Architecture générale d'une unité de traitement (Binaire)

Notation : - TA, TB et TC sont des registres tampons,
- M, est une mémoire type RAM,
- UAL, est l'unité arithmétique et logique
- C, C1, ...,C86 sont les commandes issues de la partie contrôle.

Exemple :

Soit à réaliser l'opération [(M2) + 1→ M1] dans l'unité de traitement de la figure(28); cette opération consiste à incrémenter le contenu de la case mémoire M2 et à stocker le résultat dans la case mémoire M2.

Cette opération se déroule en plusieurs étapes dans l'unité de traitement de la figure (28) et selon l'ordre suivant :

- lecture de la donnée (M2) :
  - . Sélection de la case mémoire M2 par (C17, C19,..., C33),
  - . Lire le contenu de M2.

- transfert de la valeur (M2) dans TA :
  - . Transfert entre la mémoire M et le tampon TA (C4 = 1),
  - . Mémorisation de (M2) dans TA.

- calcul de la valeur ((TA) +1) dans l'UAL :
  - . Sélection de la fonction "+1" de l'UAL par (C52, C71, ..., C86)
  - . Calcul de la valeur((TA) + 1)

- transfert de la valeur ((TA) + 1) dans TC :
  - . Transfert entre l'UAL et TC (C7 = 1),
  - . Mémorisation de ((TA) + 1) dans TC

- transfert du contenu de TC sur le bus-DB :
  - . Transfert entre TC et DB (C3 = 1) ;
  - . Connection du tampon TC sur le bus-DB.

- Transfert de la valeur (TC) dans M1 :

  . Transfert entre DB et la mémoire M . (C1 = 1)

  . Mémorisation de la valeur (TC) dans M1 :

    . sélection de la case mémoire M1 (C17, C19, ..., C33),

    . écrire la valeur (TC) dans M1

La valeur ((M2) + 1) est dans la case mémoire M1.

Remarque :

Cette notion d'aiguillage des données, au niveau d'une unité de traitement, met en évidence une structure matérielle comme à l'ensemble des éléments de la partie opérative.

Ainsi, chaque élément de la partie opérative peut être considéré comme un ensemble de deux composantes :

- un bloc de contrôle, qui est constitué essentiellement de circuit combinatoires, joue le rôle d'une barrière.
- un bloc opératif qui est constitué d'éléments de mémorisation et d'éléments de connexion, détermine la fonction spécifique de l'élément considéré.

D'ailleurs, cette notion peut être étendue à tous les éléments que constituent un système séquentiel par exemple :

- Au niveau d'un système (carte)

  Microprocesseur/Espace mémoire

- Au niveau d'un circuit séquentiel :

  Partie contrôle/partie opérative

- Au niveau d'une partie contrôle :

  Séquenceur/Interface de commandes.

- Au niveau des portes logiques :

Barrière d'activation/place *

Si on affecte un numéro à chacun de ces niveaux, selon un certain critère de finesse de description, on peut décrire à partir des objets d'un niveau (i) tous les objets d'un niveau supérieur et ceci avec des degrés de détails et d'encombrements divers.

Ainsi, le formalisme DELTA peut être le point de départ de toute une gamme de formalisme de description des circuits séquentiels alliant l'aspect architectural et l'aspect comportemental des objets et ceci à des degrés de finesse différents.

---

* Voir définition et exemple de la notion d'ETA (PARTIE.1).

PARTIE.4

PROJET 6800.S

PRESENTATION GENERALE

Dans cette partie, nous effectuons un "zoom" sur un microprocesseur (*) standard, universel par son utilisation, qui permet de mettre en évidence quelques problèmes généraux liés à la technique de la conception manuelle.

Précisons que le microprocesseur en question est le MC 6800 et que sa conception a été réalisée par MOTOROLA de manière tout à fait manuelle. Actuellement, ce microprocesseur est fabriqué en France par la Société EFCIS.

Ce qui nous parait intéressant de souligner à travers cette étude, c'est la nécessité du traitement automatique (simulation logique du fonctionnement, optimisation de l'algorithme de la machine, ...) pour limiter les risques d'erreurs (oh, combien grands!) dûs à l'intervention humaine directe dans la conception d'un circuit séquentiel.

Comme nous le montrons dans cette étude, ces risques d'erreurs sont bien réels et peuvent engendrer de graves conséquences au niveau des applications des utilisateurs.

(*) Cette etude a ete realisée dans le cadre du contrat EDF-ENSIMAG n°511.78.1.0

Nous analysons les différentes caractéristiques du circuit aussi bien du point de vue du matériel que de celui du logiciel.

Une étude plus détaillée, réalisée à partir du décodage manuel respectivement de microphotographies du circuit integre et de son schema logique complet ,est presentée dans (29).

Le anomalies de fonctionnement du MC 6800 sont mises à jour et des améliorations incluant la correction de celles-ci, sont présentées à la fin de cette partie, ce qui permet la réalisation physique d'une version améliorée de ce microprocesseur: le 6800.S.

Une description DELTA de la partie contrôle du MC 6800 a été réalisée pour tester les capacités de description du formalisme DELTA ; cette dernière est présentée dans l'ANNEXE .1 .

VII- GENERALITES

1. - Caractéristiques techniques

Le microprocesseur 6800 est un circuit séquentiel biphase produit par MOTOROLA, EFCIS, AMI, HITACHI. Comme la plupart des microprocesseurs de sa génération (I 8080, S 2650, Z 80, I 8085; I 8748 , MC2,...) le 6800 est un circuit monolithique travaillant sur des données de 8 bits: il reste l'un des standards des 8 bits de par sa souplesse et son taux d'utilisation.

Il existe plusieurs versions du 6800 dont la différence essentielle est la fréquence de ses phases d'horloge. Notons que certains circuits monolithiques sont issus directement du 6800 et constituent une famille de microprocesseurs.

En effet, le 6802 est un microprocesseur monolithique composé du 6800, d'une RAM (128 octets) et d'une horloge interne.

Le 6801 est également composé du 6800 et de plus, il possède une ROM (2 Koctets), une RAM (128 octets), une horloge interne et 4 ports (entrée/sortie parallèles).

Nous présentons ci-dessous, les versions produites par EFCIS (référence SFF9...):

REFERENCE	FREQUENCE MAX(MHZ)
SFF9.68M00 (*)	1
SFF9.6800	1
SFF9.68A00	1.5
SFF9.68B00	2
SFF9.6802	1
SFF9.6801	1

(*) Reservée à un usage Militaire

Les masques de fabrication du circuit intégré du 6800 (donc l'architecture interne) sont les mêmes pour toutes les versions, ceci pour un même constructeur.

Cependant, des différences locales peuvent exister au niveau des masques de fabrication pour des constructeurs différents: MOTOROLA et EFCIS ont les mêmes masques.

Notons que les versions A, B et 00, fabriquées à la même période, ne diffèrent que par leur étiquette et les performances qu'elles réalisent.

En effet, des tests de performances sont effectués à la production et un circuit reçoit l'étiquette A, B ou 00, et devient 68A00, 68B00 ou 6800, selon le résultat de ceux-ci.

Cependant, les masques peuvent evoluer avec le temps pour l'amelioration dudu produit et d'eventuelles modifications sont possibles.

Ainsi, les circuits intégrés 68A00, antérieurs à 1977, diffèrent des circuits 68A00 actuels :

- les premières versions (avant 1977) sont réalisées à partir de la technologie MOS canal N, MOS de charge à enrichissement(grille silicium),
- les versions actuelles quant à elles sont réalisées à partir de la technologie MOS canal N, MOS de charge à depletion (Grille silicium).

2. - Jeu d'instructions

Nous allons survoler les caractéristiques principales du jeu d'instructions du microprocesseur 6800 (ainsi que celles de sa famille) pour mieux saisir les particularités de son architecture interne.

Le jeu d'instructions du 6800 comprend 72 instructions différentes qui réalisent les fonctions générales suivantes :

- arithmétique binaire et décimale,
- logique,
- décalage simple (arithmétique et logique),
- décalage circulaire,
- chargement registres,
- stockage registres,
- branchement conditionnel et inconditionnel,
- manipulation de la pile,
- traitement des interruptions.

Chaque instruction possède un ou plusieurs modes d'adressage. Le 6800 possède 7 modes d'adressage qui sont :

ADRESSAGE	FORMAT INST.	NB OCTET	ADRESSE / OPERANDE
ACCUMULATEUR	(CO)	1	Accumulateur(A,B)
IMPLICITE	(CO)	1	Sans operande explicite
IMMEDIAT	(CO) (VO)	2	(VO) etant l'operande
RELATIF	(CO) (d)	2	AB= (PC±d)
INDEXE	(CO) (d)	2	AE= (X+d)
DIRECT	(CO) (a_1)	2	AE= (00.a_1)
ETENDU	(CO) (a_h) (a_1)	3	AE= (a_h.a_1)

Ces différents modes d'adressage sont d'un usage très général et présentent des caractéristiques intéressantes quant à leur souplesse d'utilisation.

Notons qu'à l'heure actuelle, le mode d'adressage le plus développé et le plus performant dans la gamme des microprocesseurs 8 bits, est celui du 6809 dont l'architecture interne est très proche de celle du 6800, leur constructeur étant le même: MOTOROLA.

Dans un but d'optimisation du câblage de la partie contrôle, les instructions du 6800 ont été organisées en six familles (Figure 29).

Chaque famille contient un certain nombre de groupes dont les instructions sont les mêmes mais un adressage different .

Chaque groupe contient un ensemble d'instructions qui ont des caractéristiques communes (même mode d'adressage, opérant sur les mêmes accumulateurs, ...).

Ce type d'organisation nécessite une structuration du code opération "par champ ". Le jeu d'instructions du 6800 est dit orthogonal car il possède deux étapes d'interprétation. Aussi les huit bits du code opération sont divisés en deux champs (Figure 29) :

- le premier champ (codage horizontal: 4 bits de poids fort) sert à déterminer les groupes avec leurs modes d'adressage ;
- le second champ (codage vertical: 4 bits, de poids faible) sert à déterminer les instructions d'un même groupe et éventuellement, un code invalide.

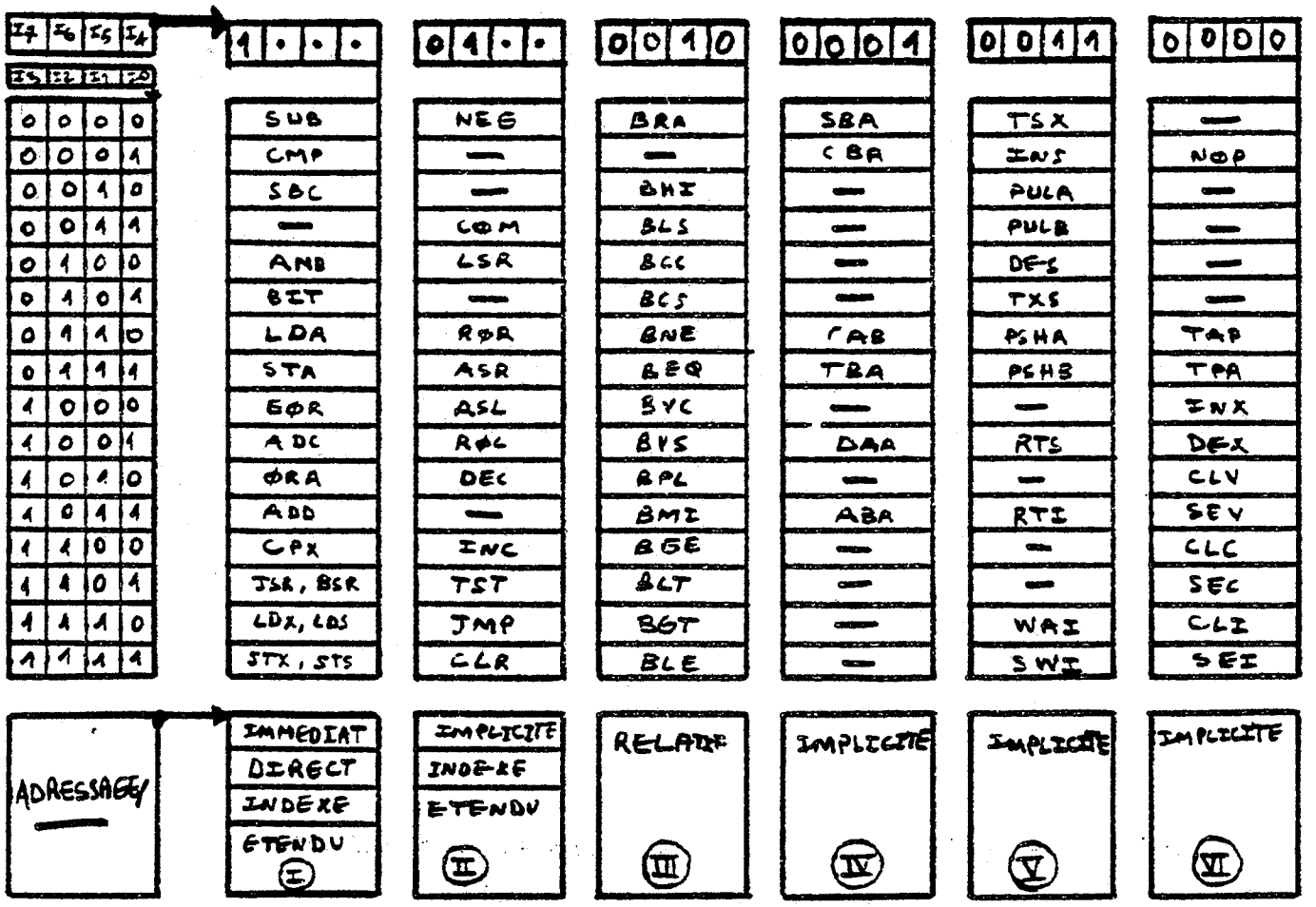
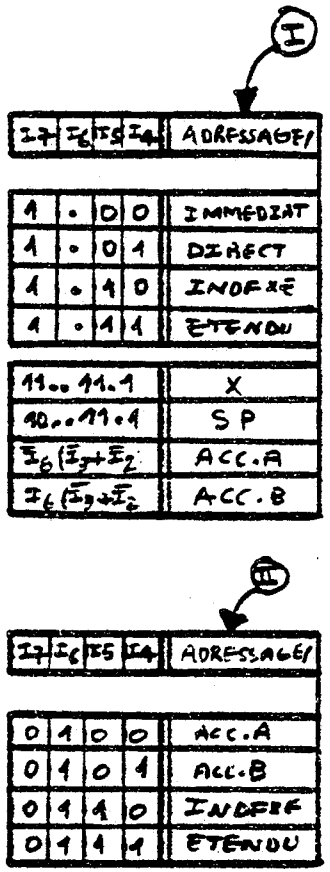


FIG.29 - Codage vertical (valable pour toutes les versions).



Codage horizontal

Le nombre de codes nécessaires à l'identification de toutes les instructions est de 197.

Le nombre de codes invalides (*) est de 59 (Figure 30)..

Comme on peut l'observer sur les deux figures suivantes, les codes invalides du 6800 n'ont pas été choisis dans un but précis comme, par exemple, pour être utilisés dans des mécanismes de détection de pannes matérielles.

La position des codes invalides dans un groupe résulte du fait qu'on ne s'est intéressé qu'au choix des codes valides permettant une simplification du câblage de la partie contrôle, les emplacements restants constituent les codes invalides. Ceux-ci peuvent être pris en compte et utilisés efficacement (13).

Cette technique de codage des instructions du 6800 est la même pour toutes les versions.

(*) Code operation qui ne definit pas une instruction du 6800.

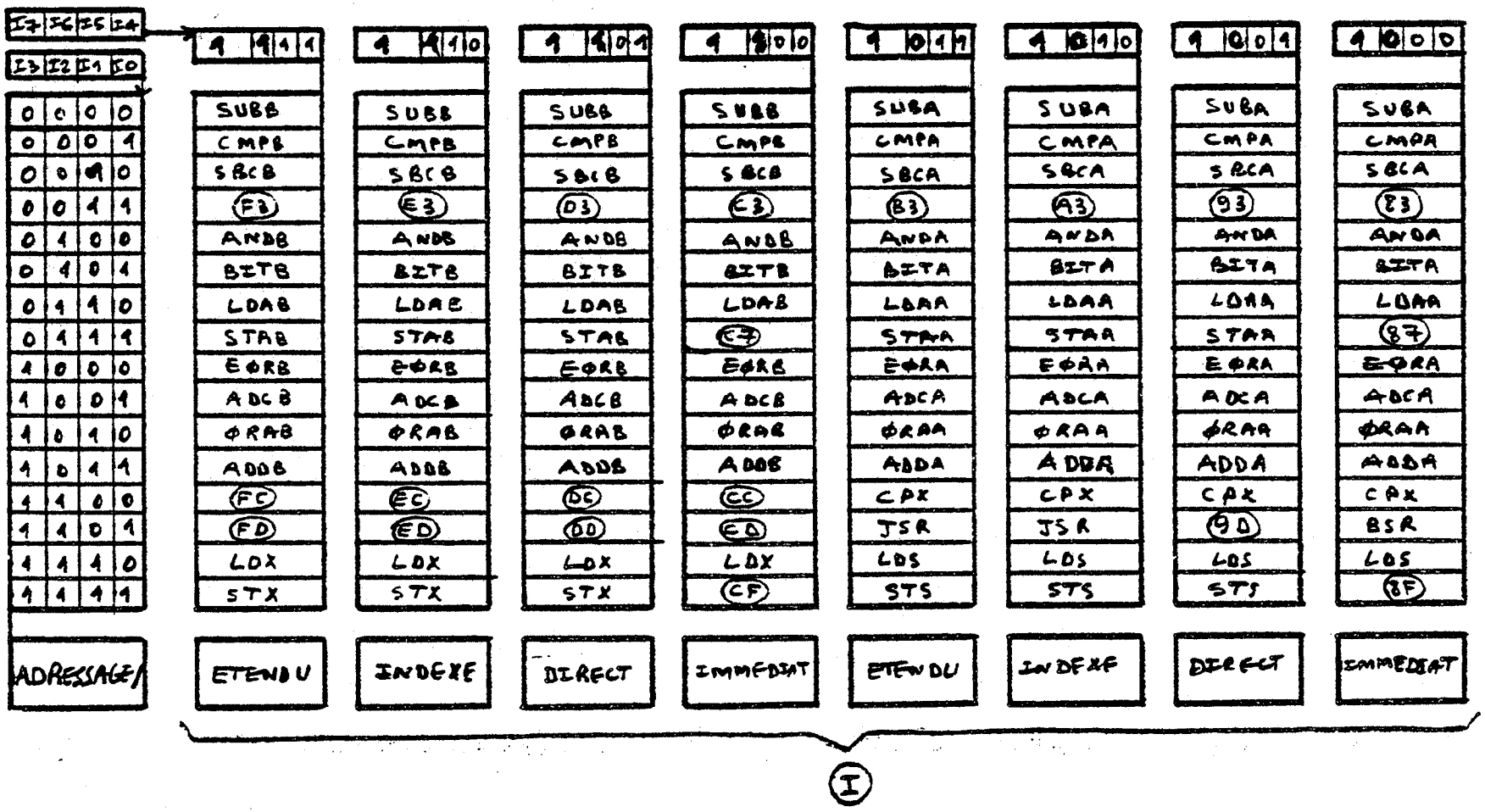


FIG.30A - Codage des instructions du 6800 (valable pour toutes les versions) .

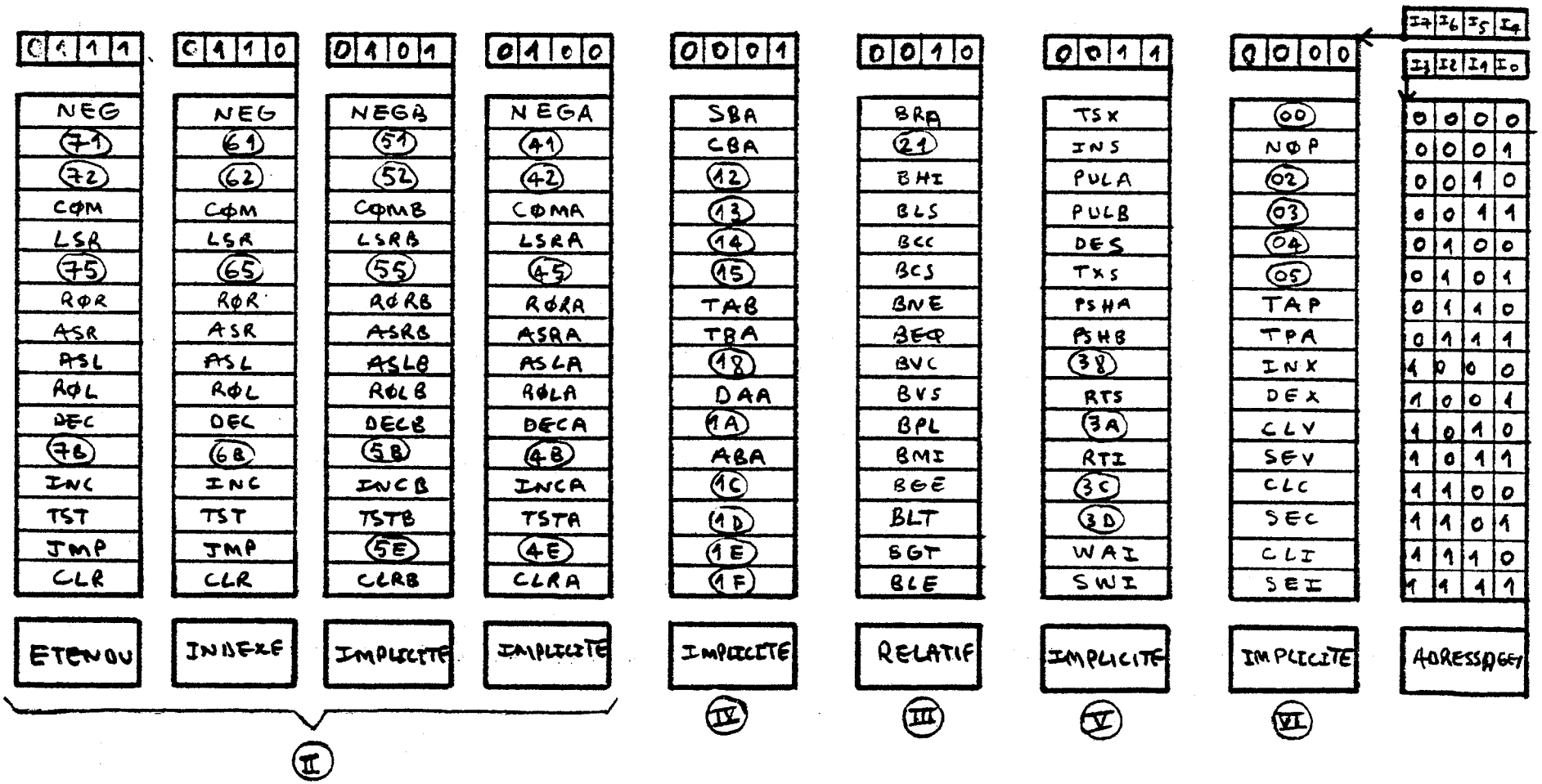


FIG.30B- Codage des instructions du 6800 (valable pour toutes les versions) / suite

VIII- ARCHITECTURE INTERNE

1. - Partie opérative

1.1. - Architecture de base

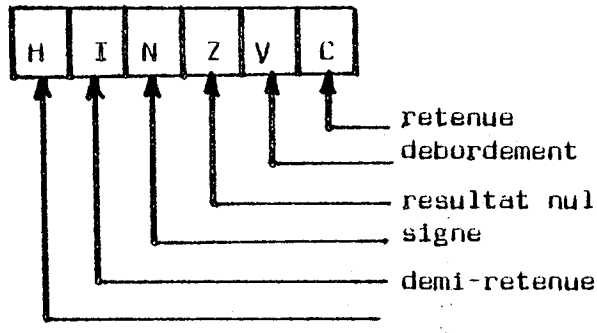
Le microprocesseur 6800 possède 14 registres internes de largeur variable (4, 6, 8 et 16 bits).

Seuls sept registres sont accessibles par l'utilisateur, les sept autres servent pour le fonctionnement interne.

Les registres accessibles par l'utilisateur sont :

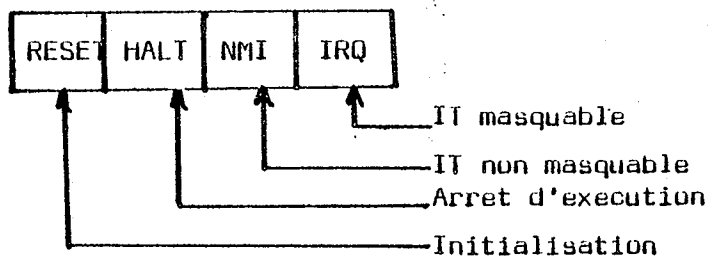
- un pointeur de programme PC (16 bits)
- un pointeur de pile ---- SP (16 bits)
- un index ---------- X (16 bits)
- deux accumulateurs ----- A,B (8 bits)
- un registre d'ETAT ----- RCC (6 bits)

Les bits C,V,N,I sont accessibles par l'utilisateur separement (instructions speciales).



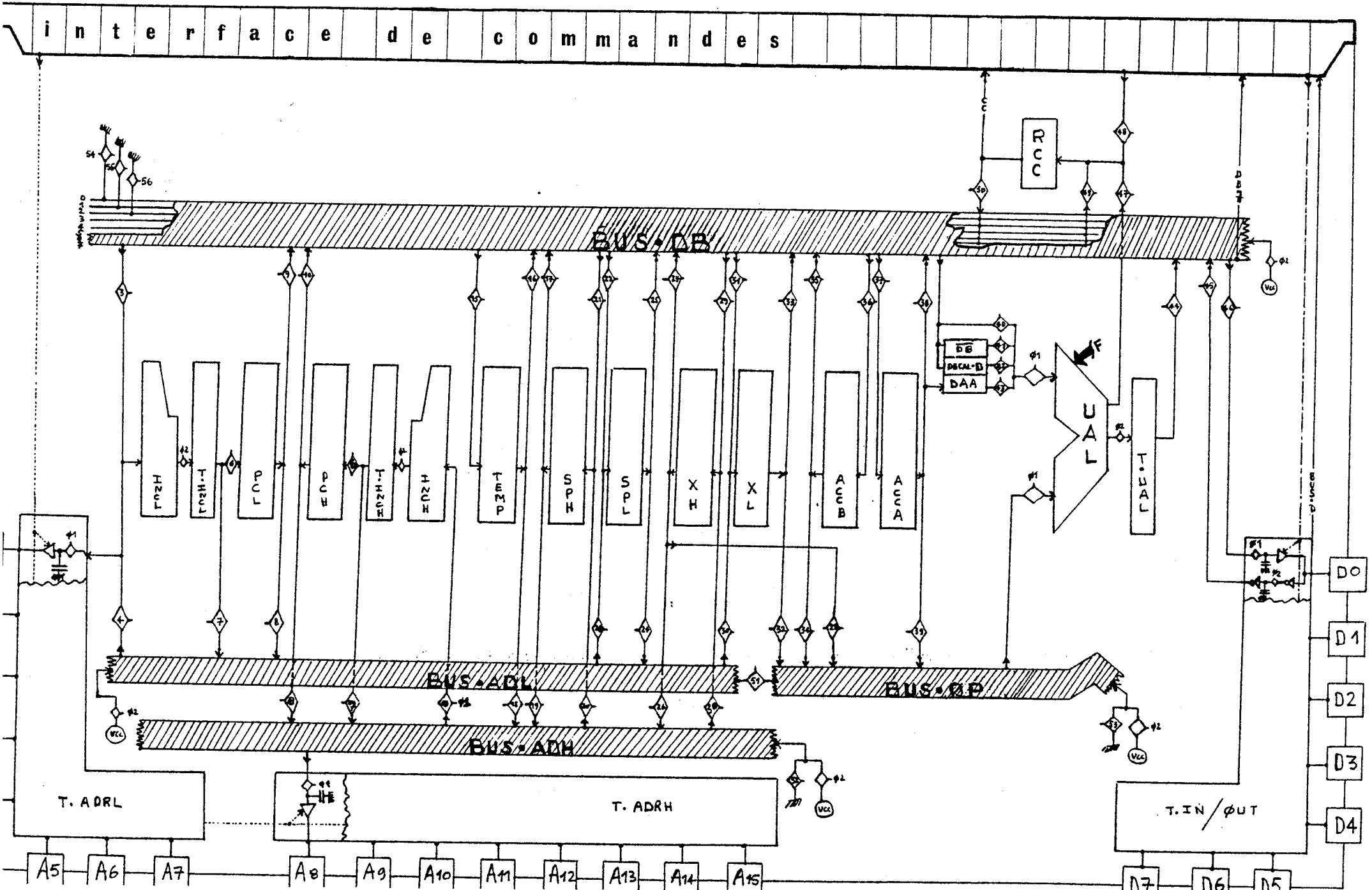
- un registre d'interruptions matérielles (4 bits)

Chacun de ces 4 bits est accessible par l'utilisateur separement (signaux d'IT)



Il n'existe pas de notion de mémoire locale de travail dans le 6800, comme c'est le cas pour la plupart des microprocesseurs 8 bits (2650, I 8080, Z 80, I 8085, ...), aussi le contrôle de ces différents registres se fait séparément pour chacun d'eux (Figure 31).

Fig.31: Partie opérative du 6800 (valable pour toutes les versions)



1.2. - Mécanisme d'exécution pipe-line :

L'analyse de l'architecture interne de la partie opérative nous montre l'existence de deux unités de traitement conçues pour travailler en parallèle.

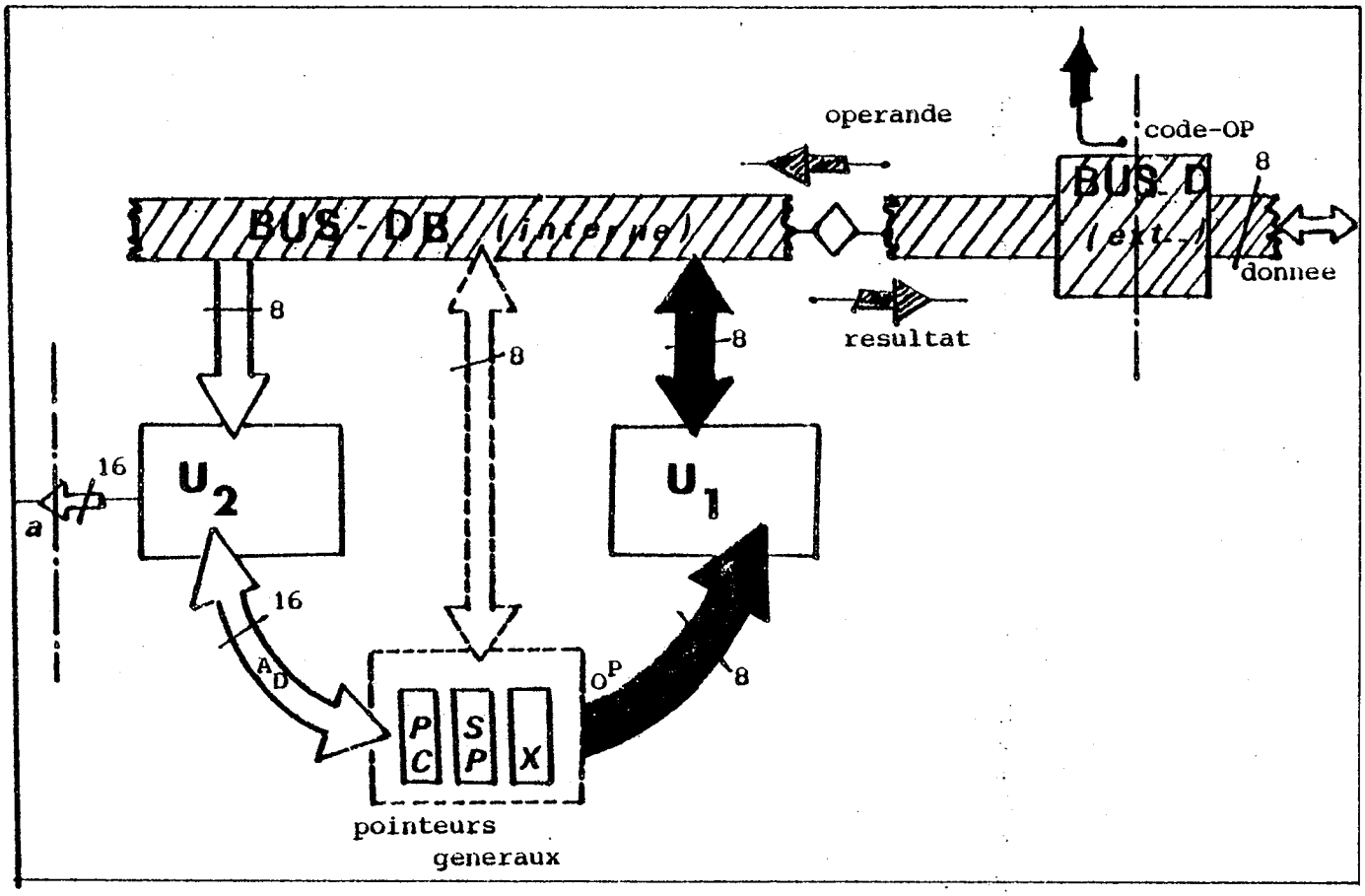


FIG.32 - Mécanisme d'exécution pipe-line du 6800.

La première unité U1 est spécialisée dans le traitement des opérations arithmétiques et logiques (largeur des données: 8 bits).

C'est une unité de traitement binaire dont les éléments sont organisés autour de deux bus (bus-DB, bus-OP).

Les éléments de l'unité U1 sont :

- l'UAL et son tampon dynamique T.UAL,
- les deux accumulateurs ACCA, ACCB,
- le circuit DAA (calcul décimal) et le registre d'état RCC.

L'unité U1 participe également à l'élaboration de l'adresse pour les modes d'adressage index et relatif. En effet, le traitement de l'opération des octets de poids faible de l'adresse (X +d) ou (PC +d) est réalisé dans l'UAL de l'unité U1. Le traitement des octets de poids fort (correction de la retenue) se fait dans l'unité U2.

La deuxième unité U2 de la partie opérative est spécialisée dans le traitement des adresses (largeur des données: 16 bits).

Il s'agit là d'une unité de traitement unaire qui se décompose en deux parties :

- une partie basse,
- une partie haute.

La partie basse qui est alimentée par l'un ou l'autre des deux bus (DB, ADL) réalise le traitement des octets de poids faible des adresses. Elle est formée d'un incrémenteur INCL (8 bits) qui réalise les opérations alimentaires (+0, +1, -1) et d'un tampon dynamique T.INCL.

La partie haute, quant à elle est alimentée par un seul bus (ADH) et réalise le traitement des octets de poids fort des adresses. Elle est également formée d'un incrémenteur incH (8 bits) qui réalise les opérations (+0, +1, -1) et d'un tampon dynamique T.INCH.

Notons que ces deux parties travaillent en parallèle et que toute adresse rentre dans l'incrémenteur (INCH, INCL) sort automatiquement sur le bus adresse externe du microprocesseur.

La liaison entre le bus de données interne (DB) et le bus de données externe (D) n'est établie que pendant la lecture des opérandes ou le stockage des résultats dans la mémoire externe.

Le bus (D) par contre, est relié directement à l'entrée du registre instructions RI (situé dans la partie contrôle), ce qui permet le chargement du code opération dans celui-ci, sans passer par le bus interne (DB).

Ceci va permettre de réduire considérablement le temps d'exécution réservé à certaines instructions.

L'exécution pite-line (*) se produit à chaque fois que l'instruction en cours possède un mode d'adressage implicite ou à accumulateur.

L'exécution d'une instruction quelconque se déroule en trois étapes et selon l'ordre suivant :

A - chargement du code opérationne et du deuxieme octet,
B - chargement de l'opérande s'il existe,
C - traitement de l'opération correspondante.

Le mécanisme d'interconnexion des bus (DB) et (D) associé à la structure de la partie opérative fait que l'opération de l'instruction en cours, qui utilise le bus (DB), est réalisée en parallèle avec le chargement de l'instruction suivante par l'intermédiaire du bus (D).

Pour mieux visualiser cette notion de parallélisme dans l'exécution de deux instructions, nous présentons le traitement (dans le temps) de trois instructions I1, I2, I3 (Figure 33)

Nous choisissons l'instruction I2 avec un mode d'adressage à accumulateurs.

(*) Exécution simultanée de deux instructions.

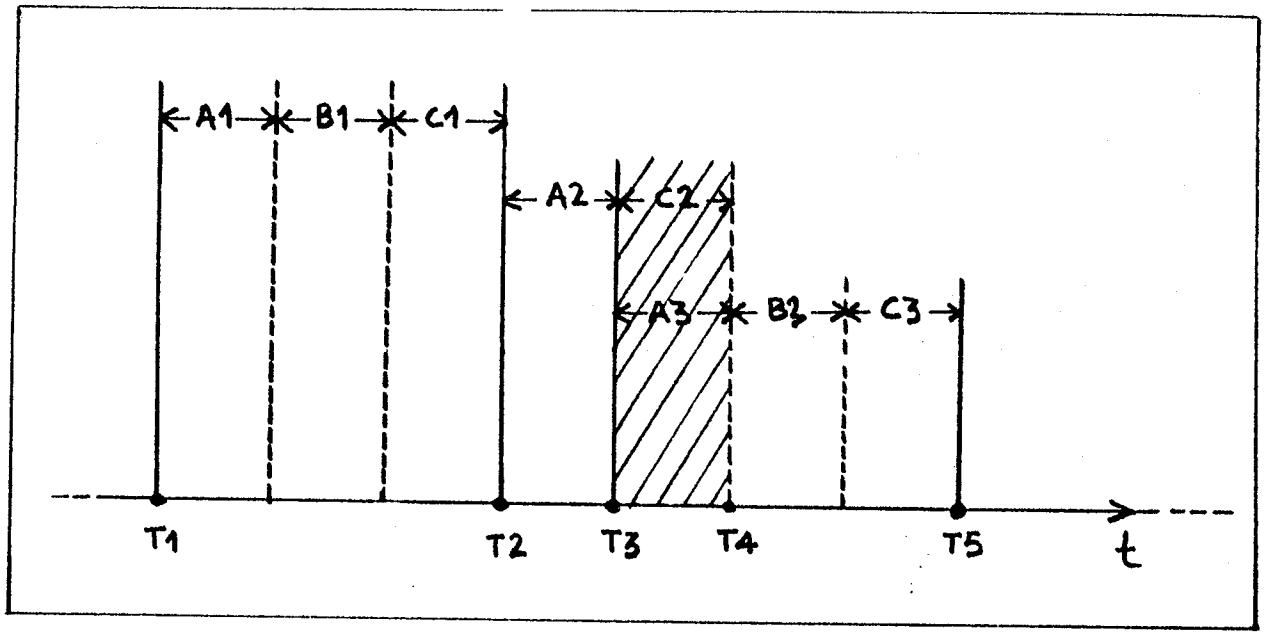


FIG.33: Execution PIPE-LINE (partie hachuree)

NOTATION:

Ai : chargement de l'instruction Ii

Bi : chargement de l'operande de Ii

Ci : operation de l'instruction Ii

Le traitement de l'instruction I1 se déroule sans parallélisme d'exécution de T1 à T2. Le chargement (A2) de l'instruction I2 a lieu de T2 à T3. Ensuite, de T3 à T4, la séquence opération (C2) de l'instruction I2 se déroule en parallèle avec le chargement (A3) de l'instruction I3.

Ceci permet un gain de temps appréciable dans l'exécution des programmes d'application, la fréquence de telles instructions étant élevée.

1.3. - Caractéristiques techniques de la partie operative

Le schéma de la figure (31) représente aussi la topologie que l'architecture interne de la partie opérative du 6800.

La disposition des registres et des différents opérateurs correspond exactement à la topologie du circuit intégré pour les versions actuelles (postérieurement à 1977).

La topologie de la partie opérative est légèrement différente sur le circuit intégré pour les premières versions (antérieures à 1977). Cependant, l'ensemble des chemins de données au niveau des deux unités de traitement est le même pour les deux types de versions.

Les registres internes sont réalisés à partir de l'un des trois points mémoire suivants, selon leur usage.

Registre dynamique :
Ce type de registre est utilisé surtout pour réaliser les tampons des deux unités de traitement tels que : T.UAL, T.INCH, T.INCL , T.IN, T.OUT, T.ADH, T.ADL. Ces registres dont l'élément de base est le point mémoire dynamique (FIG. 34) conservent l'information pendant un temps limité. Ceci entraîne des contraintes de temps sur la fréquence minimale de l'horloge ($\sim$ 100KHz pour le 6800). L'avantage de ces registres est la faible surface qu'ils occupent sur le circuit integre .

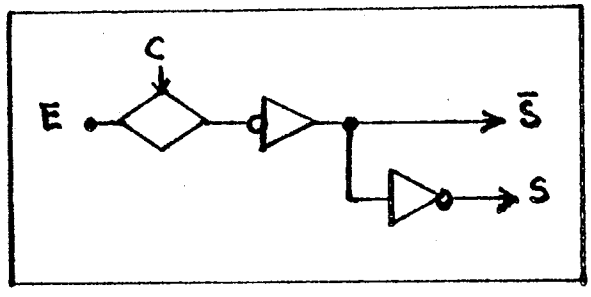


FIG. 34 - Point mémoire dynamique (avec Ampli.)

Registre statique :

Les registres statiques conservent l'information pendant un temps illimité (tant que l'alimentation fonctionne). Le principal avantage est que ce type de registre n'impose aucune contrainte sur la fréquence minimale de l'horloge. Les registres du 6800 qui sont réalisés à partir du point mémoire statique de la figure (35) et ceci pour toutes les versions actuelles, sont: ACCA, ACCB, X, SP, PC, TAMP.

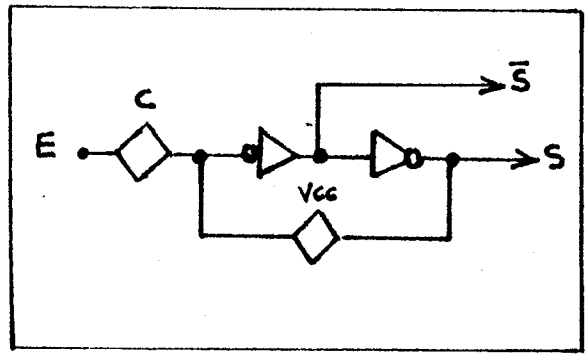


FIG.35 - Point mémoire statique

Registre semi-statique :

Ce type de registre a deux modes de fonctionnement:

- statique pendant la phase Ø2 (ou quand C.Ø1=1),
- dynamique pendant la phase Ø1 (sauf si C.Ø1=1).

Le principal avantage est la rapidité de commutation.

Le principal inconvénient (comme pour un registre statique) est la grande surface qu'il occupe sur le circuit intégré. Les registres du 6800 qui sont réalisés à partir du point mémoire semi-statique (figure 36) sont : ACCA, ACCB, X, SP, PC, TAMP, RCC- RI (premières versions) RCC, RI (versions actuelles).

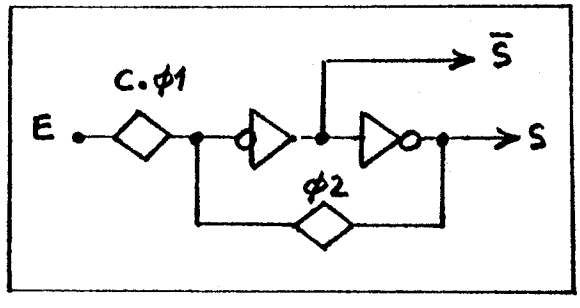


FIG.36 - Point mémoire semi-statique

2. - Partie contrôle

2.1. - Architecture de base

Le décodage manuel du circuit intégré nous a permis de comprendre le fonctionnement interne de la partie contrôle.

Le synoptique de la figure (37) représente, d'une manière précise, l'architecture interne du séquenceur avec le système d'interruption et ses deux boucles d'attente :

- attente programmée pour l'instruction WAI,
- attente câblée pour le signal HALT.

Nous distinguons essentiellement quatre blocs fonctionnels distincts au niveau du séquenceur :

- bloc (acquisitions),
- bloc (adressage),
- bloc (opération-I),
- bloc (opération-II).

Les liaisons fonctionnelles entre ces différents blocs sont représentées assez fidèlement sur le synoptique de la partie contrôle.

Pour une question de clarté, l'interface de commande n'a pas été détaillée (partie hachurée).

Toutes les commandes de la partie opérative sont représentées en détail et dans leur ordre topologique.

Notons que la partie controle du 6800 est du type machine de Mealy qui est entierement dynamique.

La notion de logique positive et de logique negative y est exploitée aussi bien au niveau du sequenceur que dans l'interface de commandes.

Les conventions adoptées sont:

Pour le sequenceur,

.Une logique positive en 01

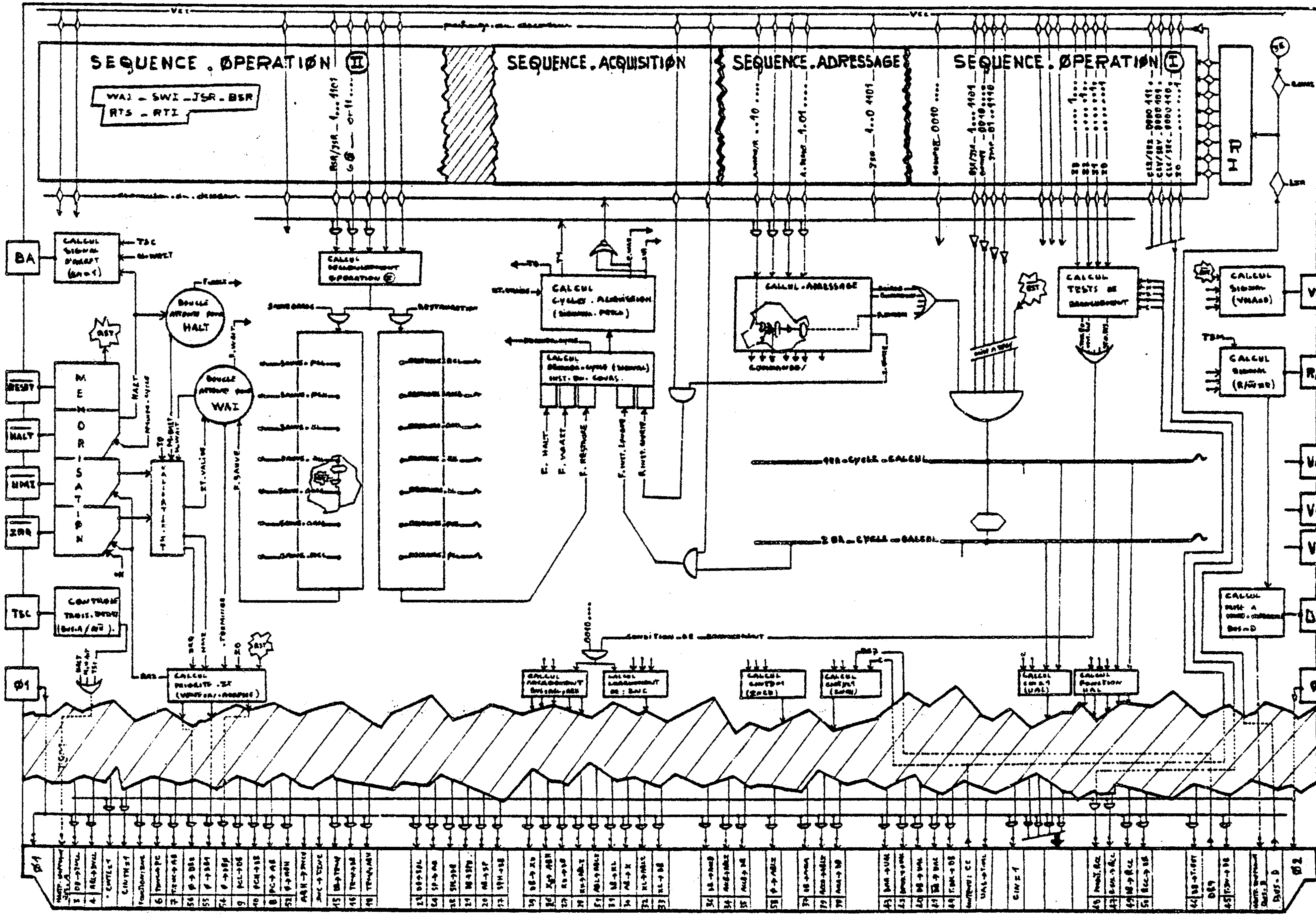
.Une logique negative en 02

Pour l'interface de commandes,

.Une logique negative en 01

.Une logique positive en 02

Fig.37: Synoptique de la partie controle du 6800 (valable pour toutes les versions)



2.2. - Fonctionnement du séquenceur :

La structure générale du séquenceur traduit la forme générale du squelette de l'organigramme de fonctionnement (Figure 38).

L'exécution de chaque instruction se déroule toujours suivant une succession d'étapes ordonnées :

- chargement du code instruction (séquence ACQUISITION),
- chargement de l'opérande s'il existe (séquence ADRESSAGE),
- exécution de l'opération correspondante (séquence OPERATION I/II).

Chacune de ces étapes est contrôlée par l'un des quatre blocs du séquenceur.

Trois blocs servent au contrôle de l'exécution des instructions arithmétiques et logiques.

Le quatrième bloc (OPERATION.II) sert au contrôle du traitement des interruptions ($\overline{NMI}$, $\overline{IRQ}$, $\overline{HALT}$, $\overline{RESET}$) et de quelques instructions de branchement (WAI, SWI, JSR, BSR, RTS, RTI).

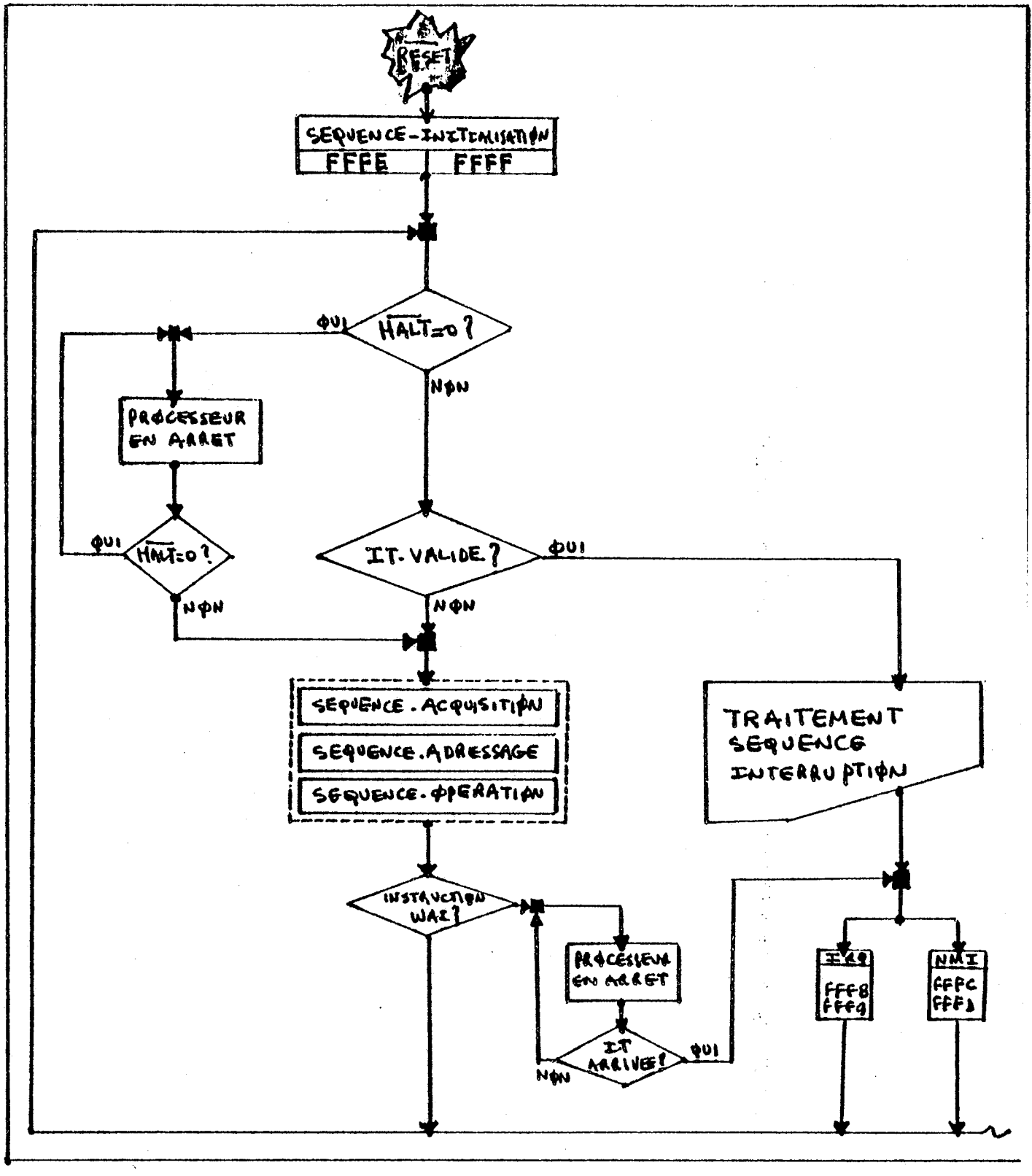


FIG. 38 - Organigramme général de fonctionnement

La partie contrôle du 6800 étant entièrement dynamique, tous les blocs du séquenceur sont realises a partir de registres à decalage dynamiques (structure: maître/esclave).

La propagation conditionnelle (figure 39) des différents signaux dans les registres à décalage se fait à travers des points mémoires dynamiques.

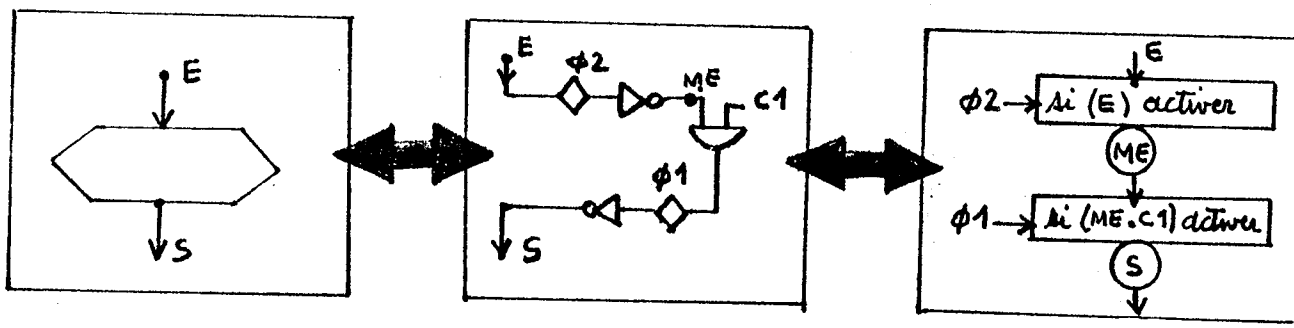


FIG. 39 - Dexcription d'un étage de registre à décalage conditionnel.

Le décodage du code opération (exécution de l'instruction en cours) est contrôlé successivement par les différents blocs du séquenceur de la manière suivante :

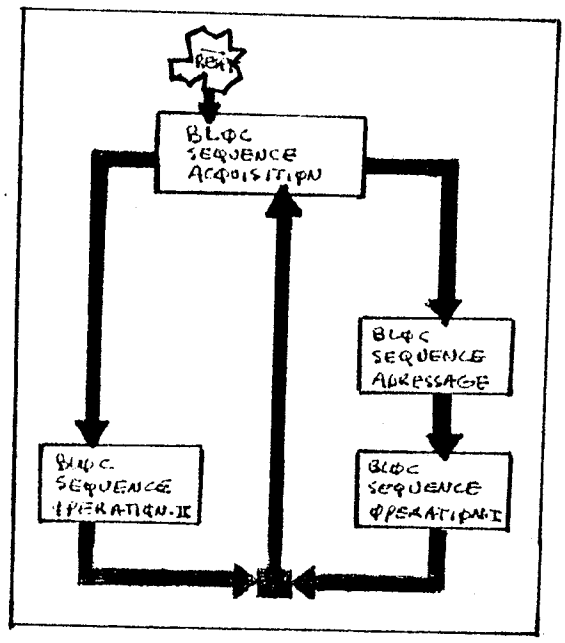


FIG.40/Structure en bloc du SEQUENCEUR

Les quatre blocs du séquenceur communiquent entre eux par la passation du contrôle d'un bloc à un autre (Figure 40).

Ainsi, à la fin de l'exécution de l'instruction RTI (par exemple), le bloc (OPERATION.II), après avoir contrôlé la restauration de tous les registres internes à partir de la pile, passe le contrôle au bloc (ACQUISITION) qui à son tour démarre le contrôle du chargement d'une nouvelle instruction.

Dans le cas général, un seul bloc est actif. Cependant, dans le cas d'une exécution pipe-line, les blocs (OPERATION.I) et (ACQUISITION) sont activés en même temps.

En effet, dans ce cas, le contrôle de l'opération de l'instruction en cours, s'effectue en parallèle avec celui du chargement de l'instruction suivante:

- le bloc (OPERATION.I) assure le contrôle de l'opération de l'instruction en cours,
- le bloc (ACQUISITION) assure le contrôle du chargement de l'instruction suivante.

Nous présentons, dans ce qui suit, une vision plus détaillée du fonctionnement des quatre blocs du séquenceur.

- Bloc (ACQUISITION)

Le bloc (ACQUISITION) contrôle entièrement l'instruction en cours pendant les deux premiers cycles. Les signaux de contrôle qui sont générés par ce bloc sont les mêmes pour toutes les instructions.

Les actions correspondant à cette séquence sont les suivantes :

- chargement du code opération au premier cycle,
- chargement du deuxième octet au deuxième cycle.

Notons que si l'instruction en cours n'a qu'un seul octet (mode d'adressage implicite ou à accumulateur), le deuxième octet est quand même chargé, mais il est perdu dans le tampon d'entrée (T.IN).

Ce même octet est rechargé une deuxième fois pour le compte de l'instruction suivante comme code opération.

Les principaux signaux de contrôle qui sont générés par ce bloc, et qui sont représentés sur le synoptique de la partie contrôle du 6800 (figure 37) sont :

T1,

est un signal qui indique le cycle numéro un de l'instruction en cours ($T1=1$). Ce signal est utilisé par le circuit (VALIDATION.IT) pour valider le traitement d'éventuelles interruptions matérielles.

LEC, WAIT,

sont deux signaux qui permettent le chargement du registre instruction RI pendant la phase Ø2 du cycle T1 ($LEC+WAIT=1$). Ils permettent également d'isoler et de précharger (*) les sorties du décodeur.

Ces deux signaux sont en mutuelle exclusion, aussi deux cas peuvent se présenter selon le signal IT.VALIDE.

(*) Mise à UN des sorties du décodeur.

1er cas : IT-VALIDE=0 (aucune IT(**) n'a été prise en compte).

(LEC=1,WAIT=0), chargement du code opération de l'instruction en dans le registre RI à partir du bus de données externe (D).

2ème cas : IT-VALIDE:1 (une IT est prise en compte).

(WAIT=1,LEC=0), chargement du code opération de l'instruction WAI(3E) par forçage. Le code opération, lu en mémoire, est perdu, il sera relu une deuxième fois après le traitement de l'IT.

est un signal qui indique le cycle numéro deux de l'instruction en cours ($T2=1$). Ce signal permet le passage de contrôle du bloc (ACQUISITION) au bloc (ADRESSAGE) ou au bloc (OPERATIO.II) selon le traitement en cours.

DERNIER-CYCLE (ADER),
est un signal qui indique le dernier cycle de l'instruction en cours. Il permet de valider l'échantillonnage du signal externe ($\overline{HALT}$).

(**) Interruption matérielle

- Bloc (ADRESSAGE)

Le bloc (ADRESSAGE) prend le contrôle au cycle $T2$. Selon le mode d'adressage de l'instruction en cours, la durée du contrôle par ce bloc est plus ou moins longue.

Soit N la durée du contrôle en nombre de cycles du CPU (*)(ce qui correspond au nombre d'étages des différents registres à décalage qui constituent le bloc (ADRESSAGE)).

On a ,

N=0 : pour le cas des modes d'adressage à accumulateur, implicite et immédiat. Le passage du contrôle à l'un des deux blocs (OPERATION.I) ou (OPERATION.II) se fait d'une manière directe à partir du bloc (ACQUISITION).

N=1 : pour le cas du mode d'adressage direct.

N=2 : pour le cas du mode d'adressage étendu et relatif.

N=3 : pour le cas du mode d'adressage indexé.

Le bloc (ADRESSAGE) se charge du contrôle de la validation de toutes les commandes nécessaires pour le chargement de l'opérande (N>=1) :

- construction de l'adresse de l'opérande,
- chargement de l'opérande.

(*) De l'anglais : Unité Centrale de traitement.

Les principaux signaux de contrôle, générés par ce bloc (Figure 37) sont :

I-COURTE,
est un signal qui indique que l'instruction en cours est du type court (instruction qui déclenche une exécution pipe-line). Ce signal permet le passage de contrôle au bloc (ACQUISITION). Ainsi, l'exécution de la séquence opération de l'instruction en cours est celle de la séquence acquisition de l'instruction suivante se déroulent en parallèle.

F-IMMEDIAT,
est un signal qui indique la fin de la séquence du mode adressage immédiat.

F-DIRECT,
est un signal qui indique la fin de la séquence du mode adressage direct.

F-INDEXE,
est un signal qui indique la fin de la séquence du mode adressage indexé.

La passation de contrôle du bloc (ADRESSAGE) au bloc (OPERATION.I) se fait par l'intermédiaire de l'un des signaux précédents.

- Bloc (OPERATION.I)

Le bloc (OPERATION.I) prend le contrôle à la fin de la séquence adressage.

Il permet le contrôle de toutes les actions nécessaires à l'exécution de l'opération de l'instruction en cours (arithmétique, logique, ...).

Les principaux signaux de contrôle, générés par ce bloc (Figure 37) sont :

1ER-CYCLE-CALCUL (CAL1),
est un signal qui désigne le cycle numéro UN de la séquence opération.

2EME-CYCLE-CALCUL (CAL2),
est un signal qui désigne le cycle numéro DEUX de la séquence opération.

F-INST-LONGUE,
est un signal qui indique que l'instruction en cours est du type long (instruction qui ne permet pas d'exécution pipe-line car le résultat de son opération doit être stocké en mémoire extérieure).

Si l'instruction en cours est du type long, le bloc (OPERATION.I) passe le contrôle au bloc (ACQUISITION) par l'intermédiaire du signal F-INST-LONGUE.

- Bloc [OPERATION.II]

Le bloc (OPERATION.II) prend le contrôle au cycle T2.
Il permet le contrôle du traitement des signaux d'interruption ($\overline{NMI}$, $\overline{IRQ}$, $\overline{HALT}$, $\overline{RESET}$) ainsi que l'exécution de la séquence opération des instructions spéciales de branchement(WAI, SWI, JSR, BSR, RTS, RTI).

Le traitement des interruptions matérielles (NMI, IRQ) se résume en l'exécution de l'instruction WAI, suivie d'un branchement vers la procédure de l'interruption correspondante. Plaçons-nous dans le cas où une IT (NMI ou IRQ) est arrivée et mémorisée.

Au cycle T1 de l'instruction en cours, le circuit [VALIDATION-IT] génère le signal (IT-VALIDE=1). Le bloc [ACQUISITION] prend en compte ce signal et génère le signal (WAIT=1) avec (LEC=0), ce qui permet le chargement du code opération de l'instruction WAI(3E) dans le registre instruction RI.

Le contrôle de l'exécution de l'instruction WAI est réalisé par le circuit [SAUVEGARDE] qui passe le contrôle au circuit [BOUCLE-ATTENTE-POUR-WAI] par l'intermédiaire du signal (F.SAUVE=1), qui à sont tour, passe directement le contrôle au bloc [ACQUISITION] et valide le circuit [CALCUL-PRIORITE.IT] par le signal (S.TERMINEE=1).
Ce dernier se charge de générer l'adresse de branchement suivant l'IT en cours.

Les principaux signaux de contrôle générés par ce bloc (Figure 37) sont :

RST,
est un signal qui permet l'initialisation du séquenceur (remise à zéro des registres à décalage des quatre blocs fonctionnels).

HALT,

est un signal qui déclenche l'arrêt de la partie contrôle (HALT=1). Le mode HALT se caractérise par la désactivation de tous les blocs et par un bouclage au niveau du circuit [BOUCLE-ATTENTE-PAR-HALT].

F.HALT,

est un signal qui indique que la fin du mode arrêt par HALT (F.HALT=1). Ce signal permet l'activation du bloc [ACQUISITION] ce qui permet le redémarrage de l'exécution.

M.WAIT,

est un signal qui indique que le séquenceur est en attente d'une interruption matérielle à la suite de l'exéuction de l'instruction WAI (M.WAI=1).

F.WAIT,

est un signal qui indique la fin de l'exécution des instructions nécessitant une sauvegarde des registres dans la Pile (F.WAIT=1).

F.SAUVE,

est un signal qui indique la fin de la séquence de sauvegarde des registres de la partie opérative dans la pile.

F.RESTAURE,

est un signal qui indique la fin de la séquence de restauration des registres de la partie opérative à partir de la pile.

S.TERMINEE,

est un signal qui indique la fin de la séquence de sauvegarde des registres pendant le traitement des interruptions matérielles et des instructions (WAI, SWI).

RAZ,
est un signal qui permet la remise à zéro de bascules d'interruption après le traitement de celles-ci (NMI, IRQ).

IT-VALIDE,
est un signal qui indique qu'une IT matérielle a été échantillonnée et mémorisée.

I,
est un signal qui désigne le masque de l'interruption IRQ (5ème bit du registre d'état : RCC).

IO,
est un signal qui désigne le bit zéro du code opération de l'instruction en cours. Ce signal est utilisé pour distinguer l'exécution des instructions WAI et SWI pendant le traitement des IT matérielles.

Le bloc [OPERATION.II] passe le contrôle au bloc [ACQUISITION] par l'un des trois signaux F.HALT, F.WAIT et F.RESTAURE suivant le traitement en cours.

2.3. - Fonctionnement de l'interface de commande

L'interface de commande qui n'est pas représentée en détail (Figure 37) utilise les caractéristiques qui sont définies au niveau du jeu d'instructions pour générer et formater les commandes qui sont destinées aussi bien à la partie opérative qu'à l'environnement extérieur.

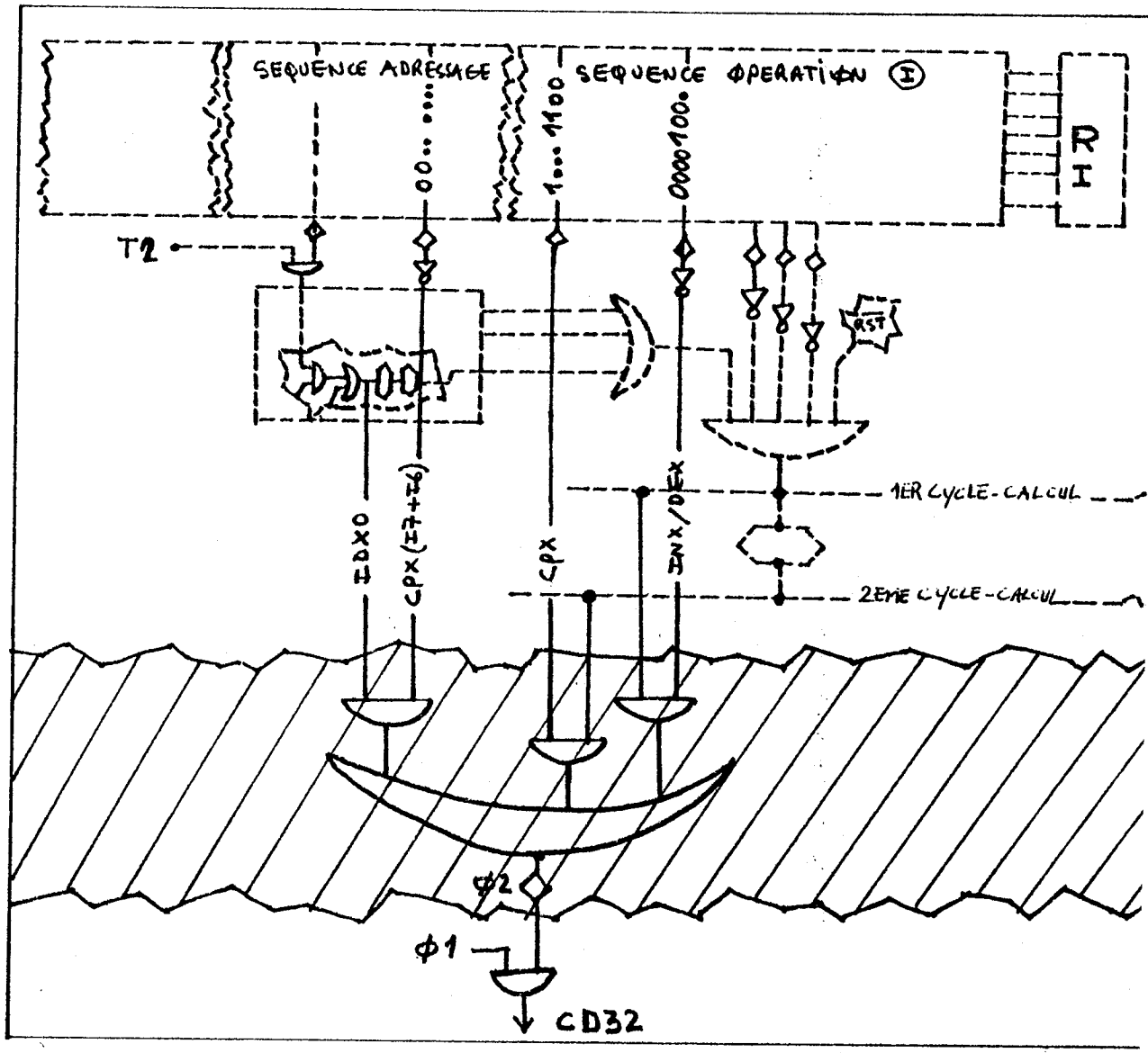


FIG. 41 - Technique de câblage dans l'interface de commande.

La génération d'une commande s'effectue à travers un circuit combinatoire (portes ET/OU) qui reçoit en entrée un ensemble de conditions extérnes (code opération, Etat de la partie operative) et valide par des signaux du Sequenceur (Machine de Mealy)

Cette technique de la génération des commandes dans l'interface de commandes est représentée sur la figure (41).

On voit sur l'exemple de ce schéma, le câblage qui permet la génération de la commande [CD32] qui correspond au transfert du contenu du registre indexé de poids faible (XL) sur le [bus-OP] de l'unite de traitement U1.

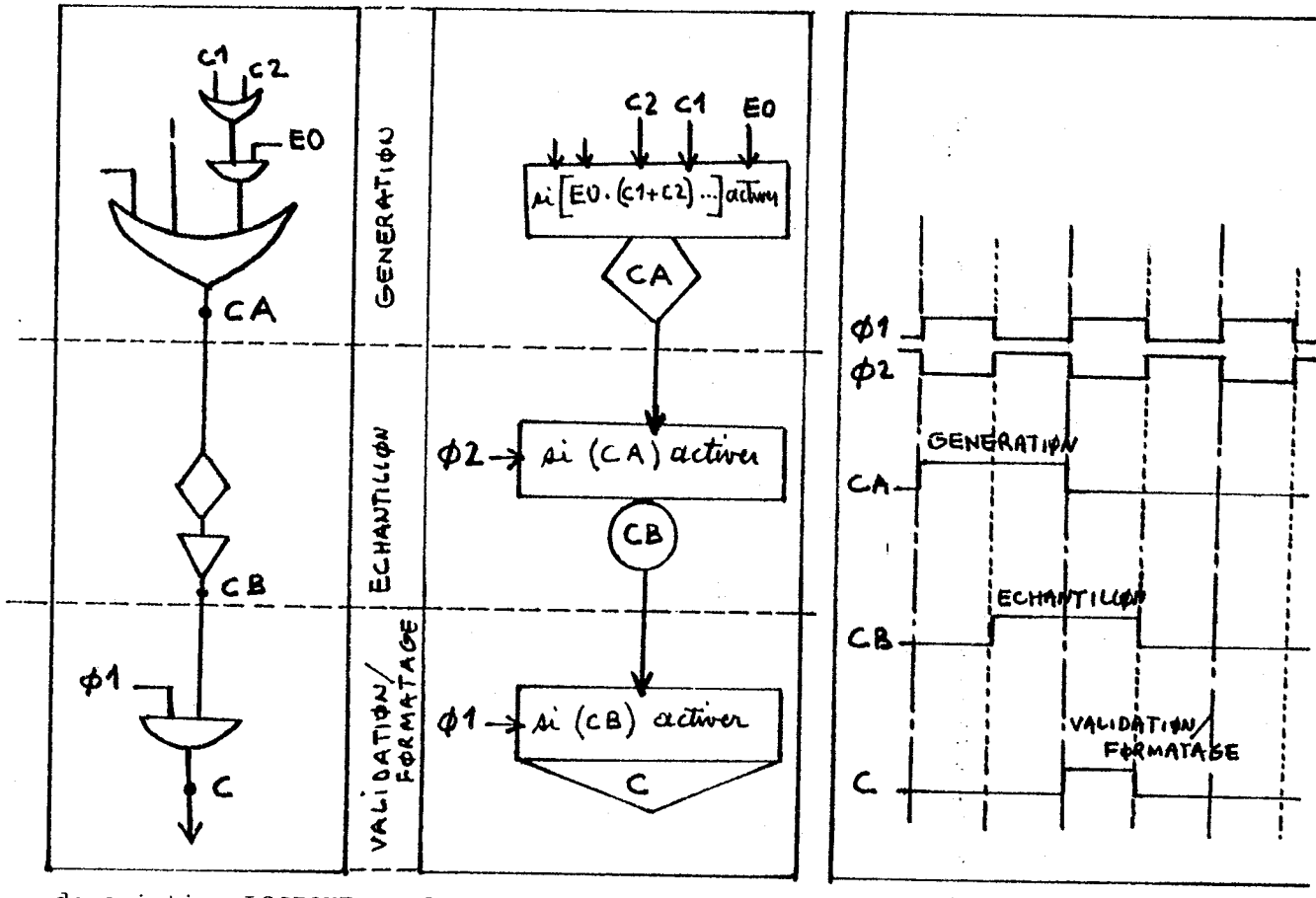
La commande [CD32] est générée par l'interface de commande dans les deux cas suivants (entre autres) :

1ER-CAS :
lors de l'exéuction de l'instruction CPX (mode d'adressage indexé) la commande [CD32] est générée pendant le premier cycle (MXU) de la séquence d'adressage et également pendant le [2ème-Cycle-CALCUL] de la séquence opération.

2EME-CAS :
lors de l'exécution des instructions (INX/DEX) pendant le [2EME-CYCLE-CALCUL] de la séquence opération.

Notons que cette commande n'est pas envoyée directement à la partie opérative, elle est d'abord retardée d'un cycle d'horloge au niveau de l'interface de commande. Ce n'est qu'au cycle suivant sa génération que cette commande devient active au niveau de la partie opérative.

Cétte caractéristique n'est pas spécifique à la commande [CD 32], elle est valable pour toutes les commandes.



description LOGIQUE description DELTA diagramme des temps

FIG. 42 - Génération des commandes dans l'interface de commande.

L'interface de commande anticipe sur la validation des actions élémentaires au niveau de la partie opérative et de l'environnement extérieur en générant les commandes un cycle avant leur activation (Figure 42)

Cette technique de câblage permet de résoudre le problème de la génération simultanée des commandes pour deux instructions différentes lors d'une exécution pipe-line.

L'anticipation dans la génération des commandes permet non seulement une réduction importante du matériel utilisé dans la réalisation d'un circuit, mais également un gain appréciable en temps d'exécution.

Notons que toutes les commandes sont validées pendant la phase Ø1 de l'horloge, sauf pour la commande [CO 46] qui est validée en Ø2 (Annexe.1).

Ceci signifie que toutes les actions (transfert, calcul, ...) qui s'effectuent dans la partie opérative, se déroulent pendant la phase Ø1.

IX - POINTS CRITIQUES DE FONCTIONNEMENT

1. - Séquence interne de démarrage à la mise sous tension

L'étude approfondie de l'architecture interne du 6800 dans les différents niveaux (logique, électrique, topologique) nous permet de décrire le comportement global du microprocesseur à la mise sous tension.

Ensuite, pour affiner la description nous procédons à la réalisation d'un test à l'aide d'une carte d'application 6800.

Cependant pour réaliser une description détaillée du comportement du 6800 à la mise sous tension, la connaissance de certaines caractéristiques matérielles de l'environnement du microprocesseur doivent être connues, telles que :

- horloge,
- source d'alimentation,
- architecture des bus externes.

Certaines précautions doivent être prises pour éviter de tomber dans l'étude de cas particuliers.

Aussi, nous choisissons un système matériel le plus souple possible pour mieux cerner les caractéristiques du microprocesseur et non celles de son environnement.

La carte d'application qui a ete utilisée pour ce test est composée des elements suivants:

- horloge (6871 A1/A2),
- microprocesseur (6800, 68 00),
- une source d'alimentation (VMAX= 5V, IMAX= 1A),
- un ensemble d'interrupteurs permettant le positionnement des differents signaux du bus de contrôle ($\overline{NMI}$, $\overline{IRQ}$, $\overline{HALT}$, $\overline{RESET}$, TSC, ...).

Le bus de donnée externe (D) est forcé à la valeur logique (00)HEXA.

L'observation du fonctionnement se fait à l'aide d'un analyseur logique et d'un oscilloscope. L'analyseur logique possède une mémoire de sauvegarde de 256 mots de 16 bits.

Cette étude a été réalisée dans deux cas différents de conditions initiales :

Dans une premier cas, la mise sous tension du microprocesseur et de l'horloge se fait en même temps (même source d'alimentation).

Danx un deuxième cas, la mise sous tension de l'horloge est faite avant celle du microprocesseur (sources d'alimentation séparées).

1.1. - Condition initiale.1

Le microprocesseur et l'horloge sont mis sous tension en même temps.

Dès que la source d'alimentation se stabilise (environ cinq volts), une phase de l'horloge (généralement Ø2) se positionne à la valeur logique '1' et l'autre phase reste à zéro. L'horloge reste bloquée dans cet état durant quelques millisecondes (∿ 50ms) ; pendant ce temps, des '1' logiques se forment sur certains étages des registres à décalages de la partie contrôle.

Pendant cette période de blocage de l'horloge (le quartz de l'horloge n'a pas encore démarré), aucune propagation des signaux ne s'effectue à l'intérieur du microprocesseur. On peut donc assimiler le microprocesseur à un grand nombre de petits circuits indépendants les uns des autres.

Les signaux de contrôle externes (VMA, R/W, BA, BUS.A, ...) prennent la valeur logique zéro. Le signal interne d'initialisation est inactif car sa valeur logique est zéro (RST=0).

Quand le quartz de l'horloge démarre, les phases Ø1 et Ø2 se debloquent et commencent à osciller correctement. Le microprocesseur commence à exécuter des instructions. La première est celle dont le code opération se trouve dans le registre instruction (RI). Pendant un temps assez court (quelques microsecondes), la partie contrôle génère des commandes désordonnées à cause des '1' logiques parasites qui se sont formés lors de la stabilisation de la source d'alimentation.

Ensuite, l'exécution des instructions se poursuit tout à fait normalement.

Dans notre système minimal, le bus de données externes (D) étant forcé à zéro, le microprocesseur charge et exécute, à chaque fois le code opération (00)HEXA qui correspond à l'instruction NOP (voir chapitre: CODES-INVALIDES).

Dans un système habituel où le bus-D est connecté à des mémoires, l'exécution dépend évidemment du contenu de la mémoire adressée. Les codes opérations (valides ou invalides) sont chargés à partir d'une première adresse (@.) de l'espace mémoire du microprocesseur. Cependant, si le contenu initial du registre instruction RI correspond à l'un des codes invalides (3C, 3D, 9D, D D), alors l'exécution se résume à une lecture séquentielle et sans arrêt de tout l'espace mémoire (voir Annexe 2). Dans notre système (BUS.D=0) ce type d'exécution est très rare pour la condition initiale.1 de mise sous tension.

Notons que la valeur de la première adresse (@.) ainsi que le contenu initial du registre instruction, ne sont pas toujours les mêmes.

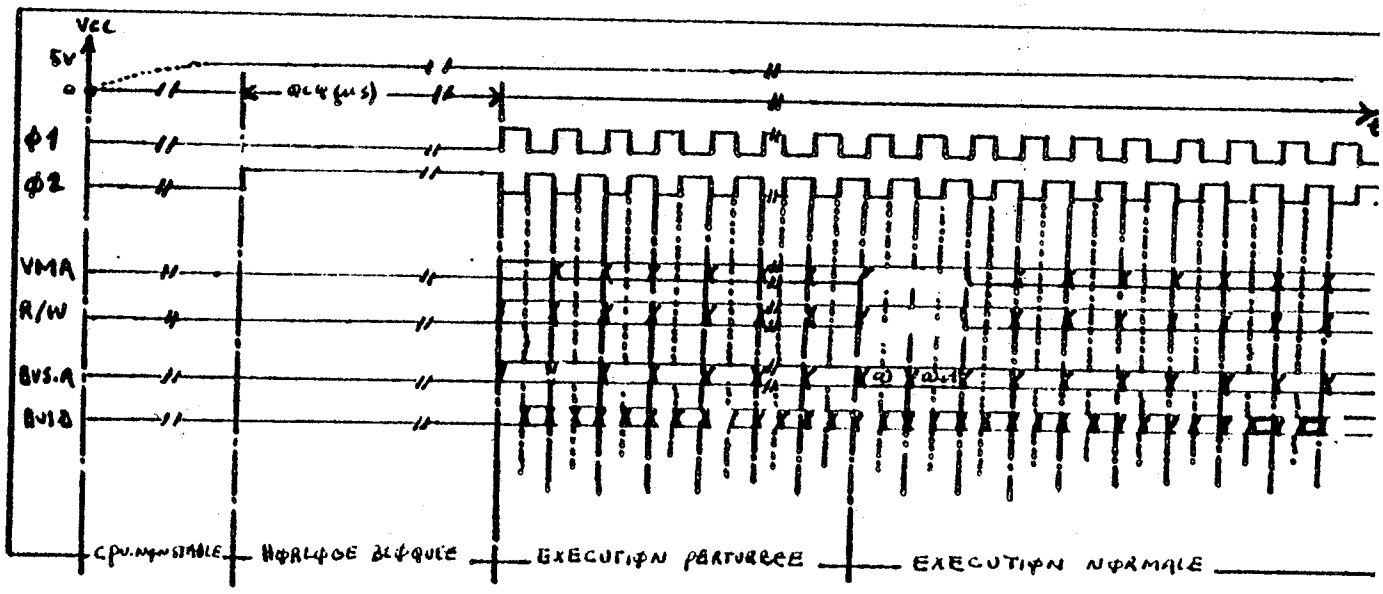


FIG. 43 - Diagramme des temps pour la mise sous tension avec la condition initiale.1 .

1.2. - Condition initiale.2

La mise sous tension de l'horloge se fait avant celle du microprocesseur.

A la mise sous tension du microprocesseur, les phases de l'horloge (Ø1, Ø2) oscillent correctement.
Pendant la période de stabilisation de la source d'alimentation, le microprocesseur se trouve dans un état indéterminé. Les valeurs logiques des commandes internes ne sont pas encore stabilisées.
Les signaux de contrôle externes sont tous au niveau logique zéro.
Arrive un moment où les commandes se stabilisent et l'exécution démarre.
Lors de la stabilisation de l'alimentation, le registre instruction se charge avec une valeur qui n'est pas toujours la même. L'exécution dépend du contenu initial du registre instruction.

Trois cas peuvent se présenter :

(a) Le contenu initial du registre instruction est l'un des codes invalides (3C, 3D, 9D, DD).
L'exécution qui suit est une lecture séquentielle et sans arrêt de l'espace mémoire du microprocesseur.
Les signaux d'interruption (NMI, IRQ, HALT) ne sont pas traités car l'exécution en cours ne se termine pas.
Ce type d'exécution est assez fréquent pour la condition initiale.2 à cause des caractéristiques electriques du registre instruction. Ceci peut être évité en créant une disymétrie électrique dans certaines bascules du registre instruction.

(b) Le contenu initial de RI est 3E.
L'exécution qui suit est donc celle du WAI. Le microprocesseur se met en attente d'interruption (BA=1, VMA=0, ...).
Le microprocesseur reste figé dans cet état jusqu'à l'arrivée d'une interruption matérielle (NMI, IRQ).

(c) Le contenu initial du registre instruction diffère des codes (3C, 3D, 9D, DD, 3E).
Le microprocesseur envoie une première adresse (@.) sur le bus adresse et démarre l'exécution des codes opérations.
L'exécution qui dépend du contenu de la mémoire adressée, se poursuit normalement jusqu'à l'arrivée du signal d'initialisation ($\overline{RESET}$ = 0).

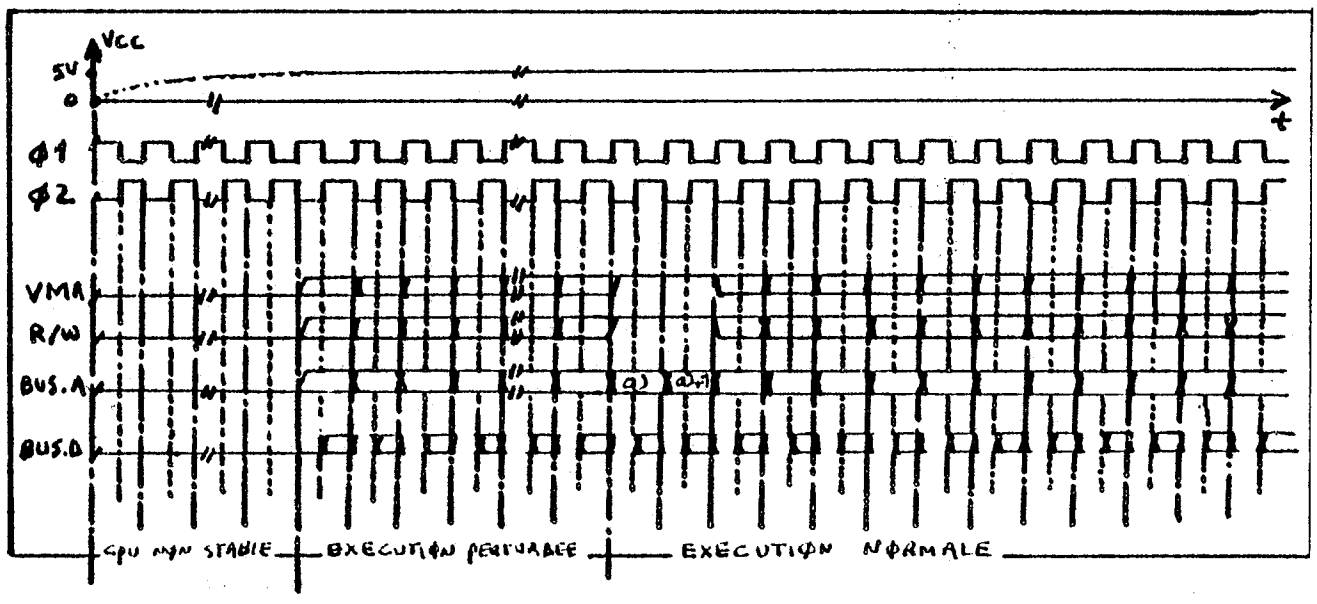


FIG. 44 - Diagramme des temps pour la mise sous tension avec la condition initiale.2.

2. - Séquence interne de démarrage lors d'une initialisation

L'initialisation du microprocesseur, soit à la mise sous tension, soit pendant l'exécution d'un programme, se fait par l'intermédiaire du signal [$\overline{\text{RESET}}$].

Dans les deux cas, la séquence d'initialisation est la même.
L'initialisation du microprocesseur ne s'effectue que sur la partie contrôle.
En effet, à l'exception du registre des codes de condition RCC (voir Chapître: Anomalies de fonctionnement), aucun autre registre n'est modifié au cours de l'initialisation [$\overline{\text{RESET}}$ = 0].
Comme nous l'avons présenté, le séquenceur de la partie contrôle est structuré en quatre blocs ; chaque bloc est constitué de registres à décalage avec une propagation conditionnelle.

L'initialisation du microprocesseur consiste à mettre à zéro tous les registres à décalages.
Les bascules de mémorisation des interruptions (NMI, IRQ) sont mises à zéro.
La mise à zéro se fait généralement sur chaque étage d'un registre à décalage.
Cependant, pour des raisons d'encombrement, la mise à zéro de certains registres à décalage se fait seulement un étage sur deux. La mise à zéro de l'étage suivant se fait par propagation du zéro de l'étage précédent.

Exemple :

Dans le bloc [OPERATION.I] (Figure 37),
seul l'étage [1ER-CYCLE-CALCUL] est mis à zéro par $\overline{RST}$,
ce qui entraîne la mise à zéro de l'étage suivant
[2EME-CYCLE-CALCUL] avec un cycle de retard.

De même, seule la partie "maître" d'un étage est initialisée, la partie "esclave" est initialisée indirectement par la propagation du zéro, avec une phase de retard.

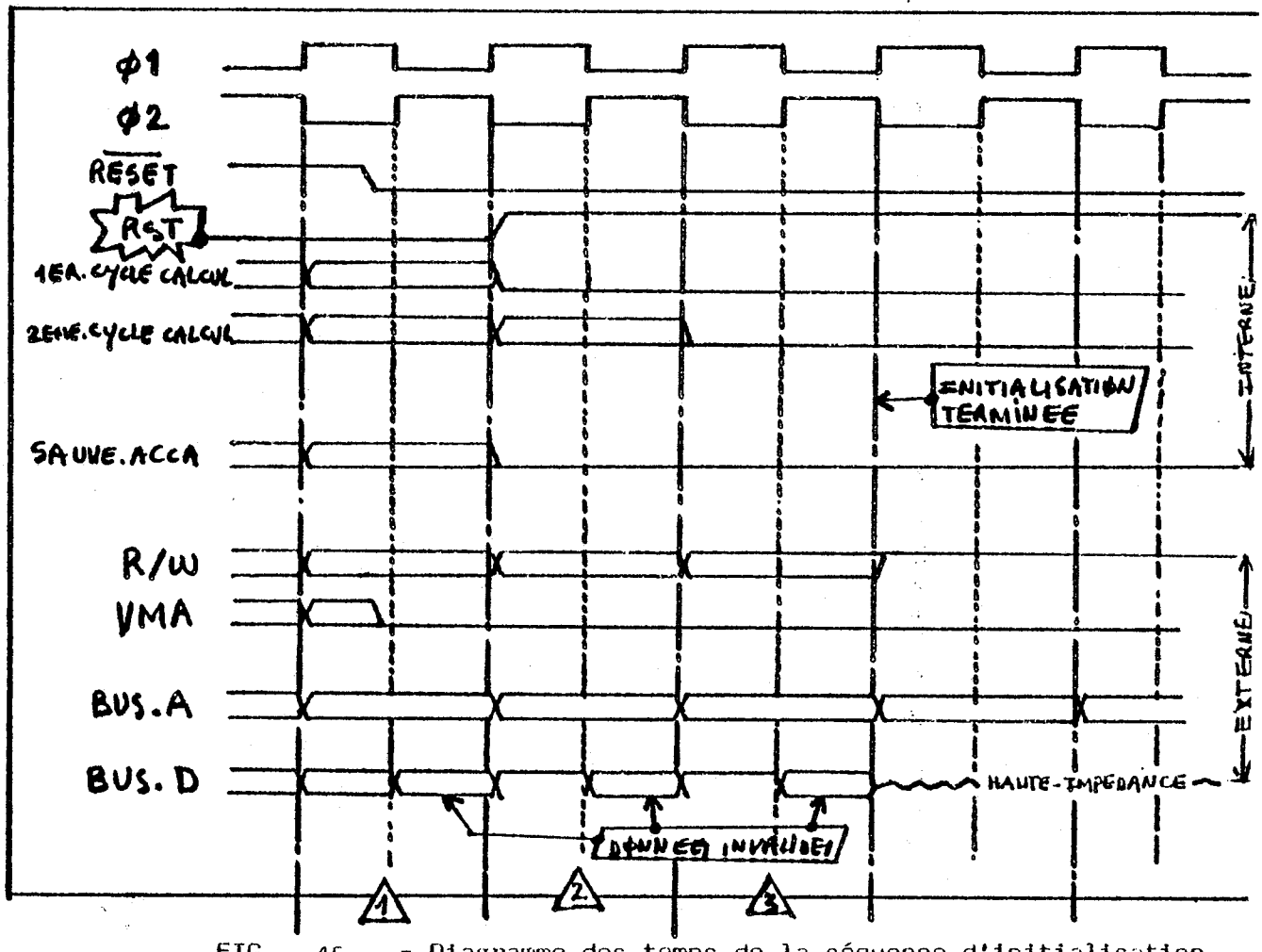


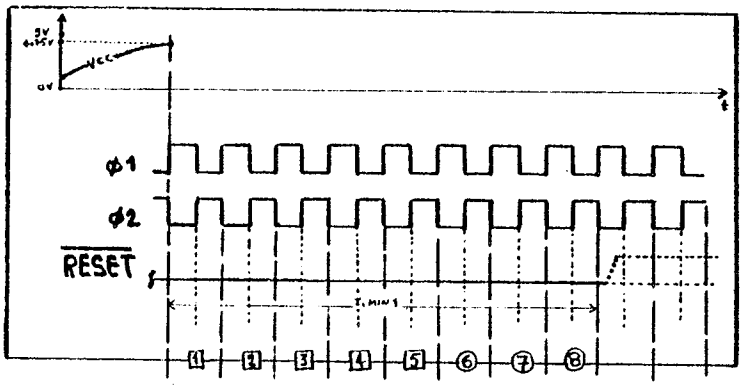
FIG. 45 - Diagramme des temps de la séquence d'initialisation.

Pour obtenir une initialisation correcte (sans parasite), le positionnement du signal [RESET = 0] doit être maintenue pendant un temps minimum T.MIN. Ce temps minimum dépend essentiellement du seuil de la tension (VCC) de la source d'alimentation.

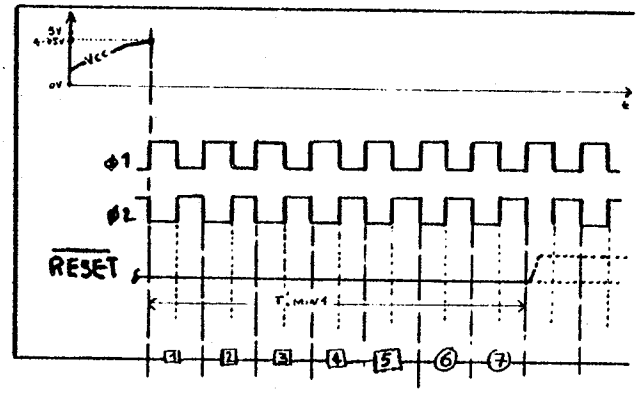
Deux cas peuvent se présenter :

- initialisation à la mise sous tension (VCC < 5 volts),
- initialisation pendant le fonctionnement normal (VCC = 5 volts).

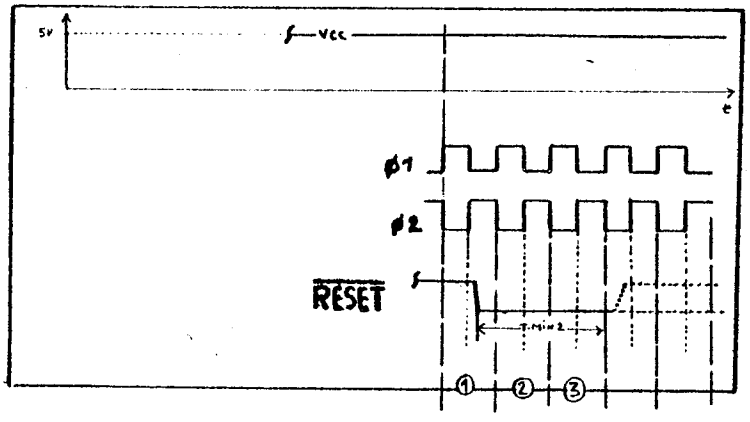
Durée minimale conseillée pour une initialisation correcte à la mise sous tension



Durée minimale obligatoire pour une initialisation correcte à la mise sous tension



Durée minimale conseillée pour une initialisation correcte pendant le fonctionnement



Durée minimale obligatoire pour une initialisation correcte pendant le fonctionnement

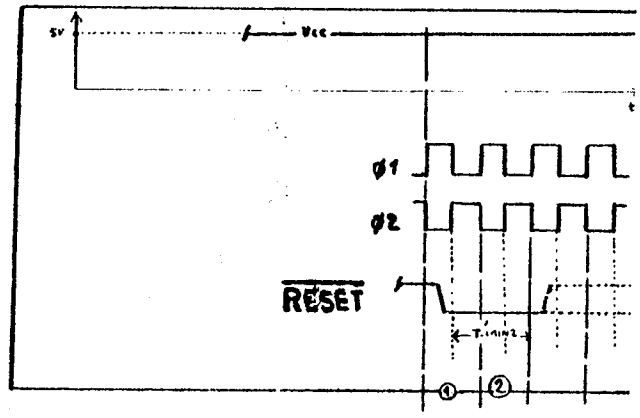


FIG. 46 - Contrainte d'initialisation du microprocesseur à partir du signal RESET.

2.1. - Initialisation à la mise sous tension

A la mise sous tension, l'alimentation du circuit intégré ainsi que celle du système global met un certain temps pour se stabiliser (amortissement des rebonds). La durée de stabilisation de l'alimentation dépend des caractéristiques électriques de chaque système.

Ainsi, pour obtenir une bonne initialisation du microprocesseur, il faut respecter les conditions suivantes :

- attendre que la ligne VCC se stabilise à 4,75 volts (au moins),
- générer le signal [$\overline{\text{RESET}}$=0] (*) et le maintenir pendant huit cycles d'horloge (au moins).

Les cinq premiers cycles permettent d'assurer la stabilisation du seuil de l'alimentation en tous points internes du microprocesseur.

Les trois autres cycles permettent d'initialiser proprement le microprocesseur. Cette durée de huit cycles peut être ramenée à sept cycles (5+2), pour des conditions particulières (Figure 46).

Si, à la mise sous tension, on attend que le seuil de la tension se stabilise à 5 volts, les conditions d'initialisation sont identiques à celles du paragraphe suivant.

(*) La génération du signal $\overline{\text{RESET}}$=0 dès la mise sous tension, serait la solution idéale.

2.2. - Initialisation pendant le fonctionnement

Le microprocesseur peut être initialisé à tout instant, même pendant l'exécution d'un programme, sauf pour les cas qui engendrent une anomalie de fonctionnement (voir Chapitre: ANOMALIE DE FONCTIONNEMENT).

Pour initialiser correctement le microprocesseur, il suffit de respecter les conditions suivantes :

- s'assurer que le seuil de la tension d'alimentation est en tous points supérieure ou égale à 4,75 volts.
- générer le signal [$\overline{\text{RESET}}$=0] et le maintenir pendant trois cycles d'horloge au moins.

Cette durée de trois cycles peut être ramenée à deux cycles, pous certaines conditions d'initialisation (Figure 46).

Si ces conditions sont vérifiées, le microprocesseur se fige dans un état déterminé (l'état initial où tous les registres à déclage de la partie contrôle sont mis à zéro).

Un fois atteint, l'état initial du microprocesseur ne change pas tant que le signal [$\overline{\text{RESET}}$=0] est présent.

3. - Traitement au niveau du système d'interruptions

3.1. - Mécanisme de validation des signaux d'interruptions

Les signaux d'interruption RESET, HALT, NMI et IRQ peuvent être envoyés au microprocesseur à n'importe quel moment, sauf pour certains cas qui engendrent des anomalies de fonctionnement (*).

Tous ces signaux sont soumis à des barrières d'échantillonnage à l'entrée du microprocesseur :

- synchronisation avec une phase d'horloge,
- masques d'interruption, ...

La prise en compte et le traitement de cas signaux, par le microprocesseur, sont plus ou moins retardés selon leur valeur et l'état interne de la partie contrôle.
La structure architecturale du système d'interruption détermine la priorité du traitement entre ces différents signaux. Le signal [$\overline{\text{RESET}}$=0] a la priorité maximale (voir séquence d'initialisation).

(*) voir chapitre: ANOMALIES DE FONCTIONNEMENT

Le signal [$\overline{HALT}$=0] a la deuxième Priorite
En effet, en l'absence du signal [$\overline{RESET}$=0] , si le signal [$\overline{HALT}$=0] est présent pendant le dernier cycle de l'instruction en cours, le microprocesseur termine l'exécution en cours et rentre dans le mode HALT (voir Figure 38) ; ceci se produit quelque soit la valeur des bascules de mémorisation des signaux d'interruption (NMI,IRQ).

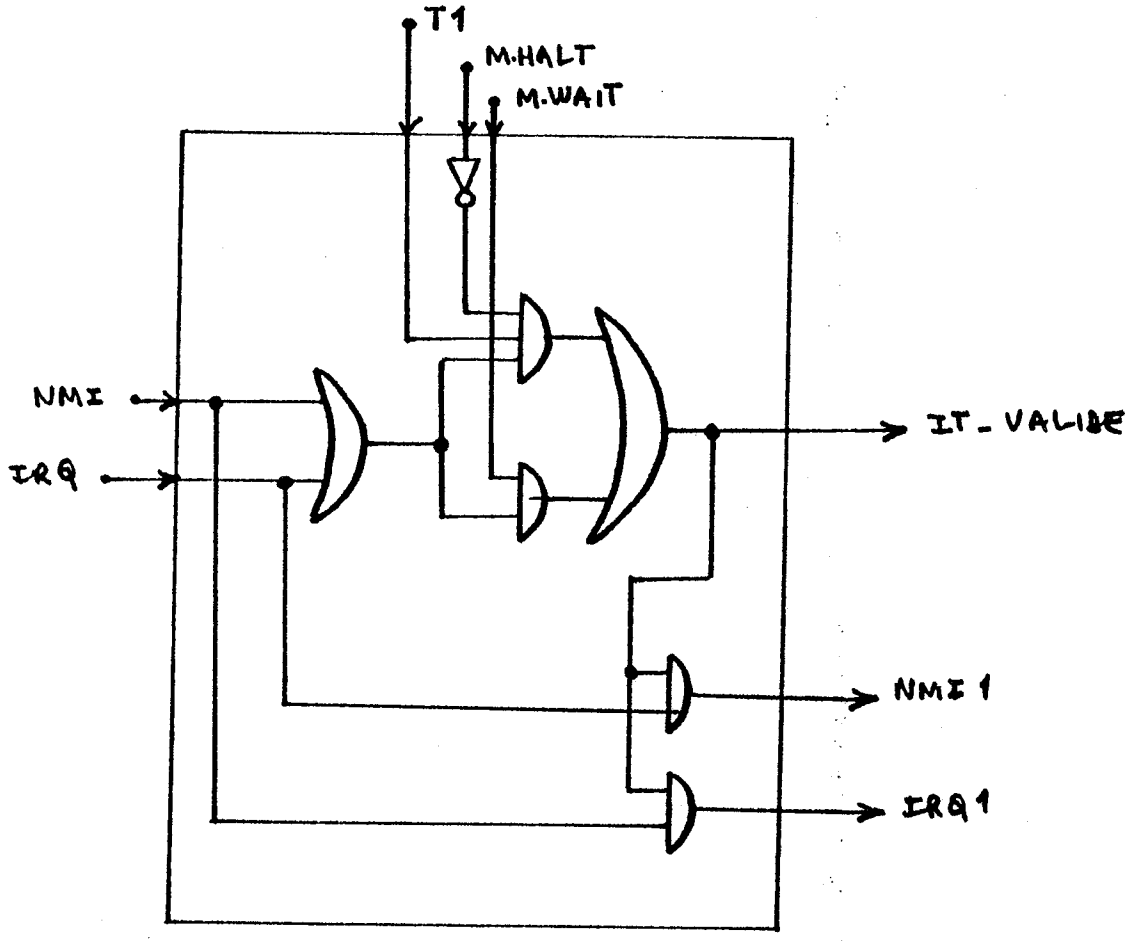


FIG. 47 - Circuit de validation des interruptions (NMI,IRQ)

Le séquenceur teste les bascules de mémorisation des signaux d'interruption (NMI,IRQ) uniquement pendant le premier cycle (T1) de l'exécution de l'instruction en cours. Notons qu'à la sortie du mode HALT, les interruptions

(NMI, IRQ) ne sont pas testées par le séquenceur, car le signal MHALT est positionné à " 1 " dans ce cas.

Cette caractéristique du 6800 permet l'exécution d'une séquence de programme sans interruption, ce qui facilite les traitements en exclusion mutuelle.

Dans le cas général, si au premier cycle d'une séquence d'acquisition, une interruption est mémorisée, alors le signal IT.VALIDE se positionne à "1" et permet au bloc [ACQUISITION] de commander le chargement de l'instruction WAI(3E) dans le registre instruction RI.

La sauvegarde du contexte (registres internes) par l'instruction WAI est commune aux deux interruptions NMI et IRQ.

A ce niveau du traitement, le séquenceur ne fait aucune distinction entre ces deux interruptions. C'est seulement pendant le cycle (10) que le circuit "CALCUL PRIORITE IT" détermine le vecteur d'adresse en fonction des signaux mémorisés. Les interruptions non masquables, (NMI) sont prioritaires sur les interruptions masquables (IRQ).

Ainsi, si le déclenchement de la séquence de sauvegarde du contexte est dû à une interruption IRQ et qu'une interruption NMI arrive avant le dixième cycle de l'exécution de l'instruction WAI, la sauvegarde du contexte continue et le vecteur d'adresse qui sera généré est celui du NMI.

Notons que l'utilisation de l'instruction WAI, dans un programme, permet d'accélérer l'accés aux programmes d'interruption . En effet, la sauvegarde du contexte étant faite par l'instruction WAI, la barrière constituée par le circuit "VALIDATION-IT" étant ouverte (M WAIT=1), l'arrivée d'une interruption permet de calculer directement le vecteur d'adresse et ainsi de commencer l'exécution du programme de l'interruption correspondante plus rapidement.

3.2. - Barrières d'échantillonnage du signal RESET

Le signal $\overline{\text{RESET}}$ permet d'initialiser l'état interne du microprocesseur aussi bien à la mise sous tension que pendant l'exécution des programmes.

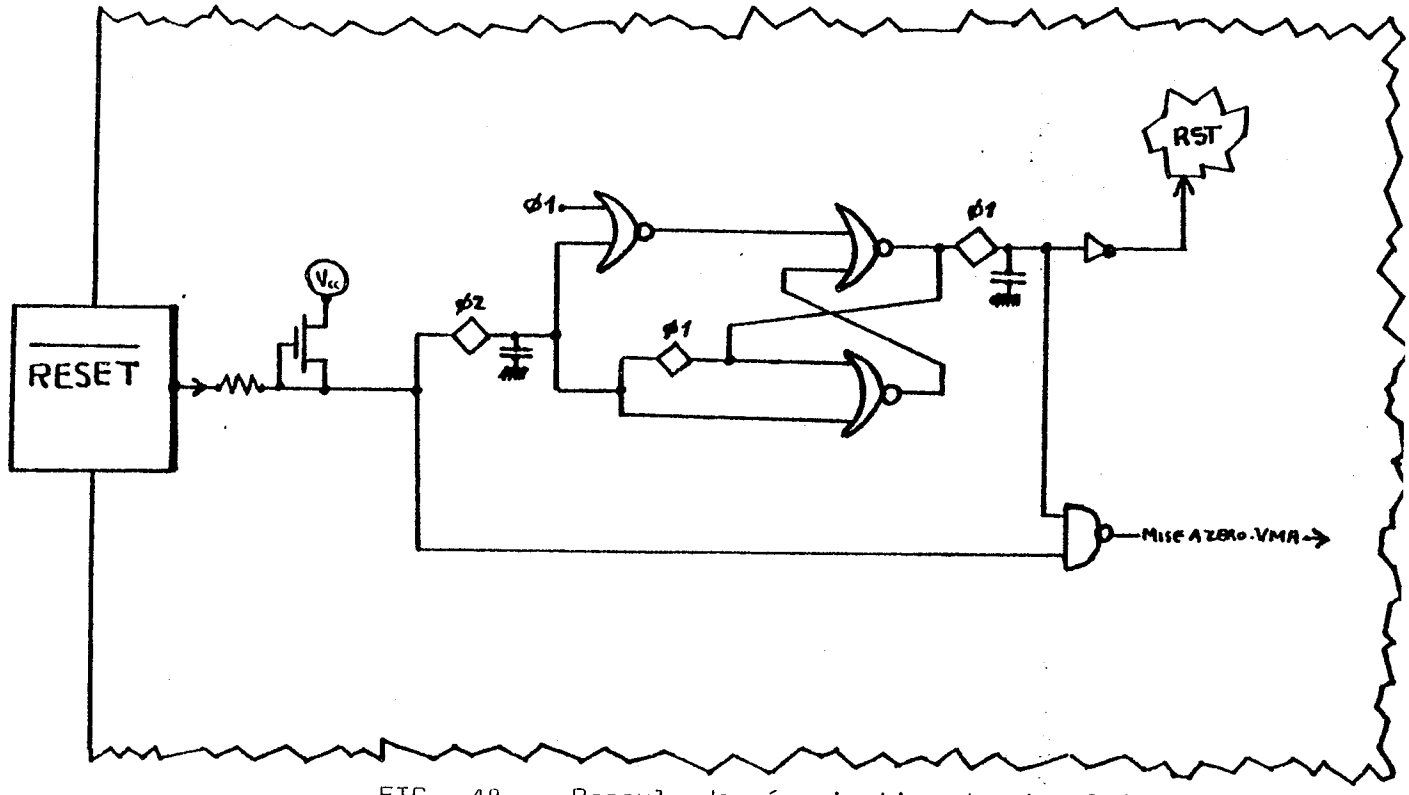


FIG. 48 - Bascule de mémorisation du signal RESET.

L'entrée [$\overline{\text{RESET}}$] du microprocesseur, possède un transistor de charge à déplétion (PULL-UP interne), aussi tout circuit à "drain-ouvert" (*) peut-il générer ce signal.

(*) Connexion de plusieurs sources à une même ligne.

Le signal externe $\overline{\text{RESET}}$, comme sa notation l'indique, est actif au niveau bas.

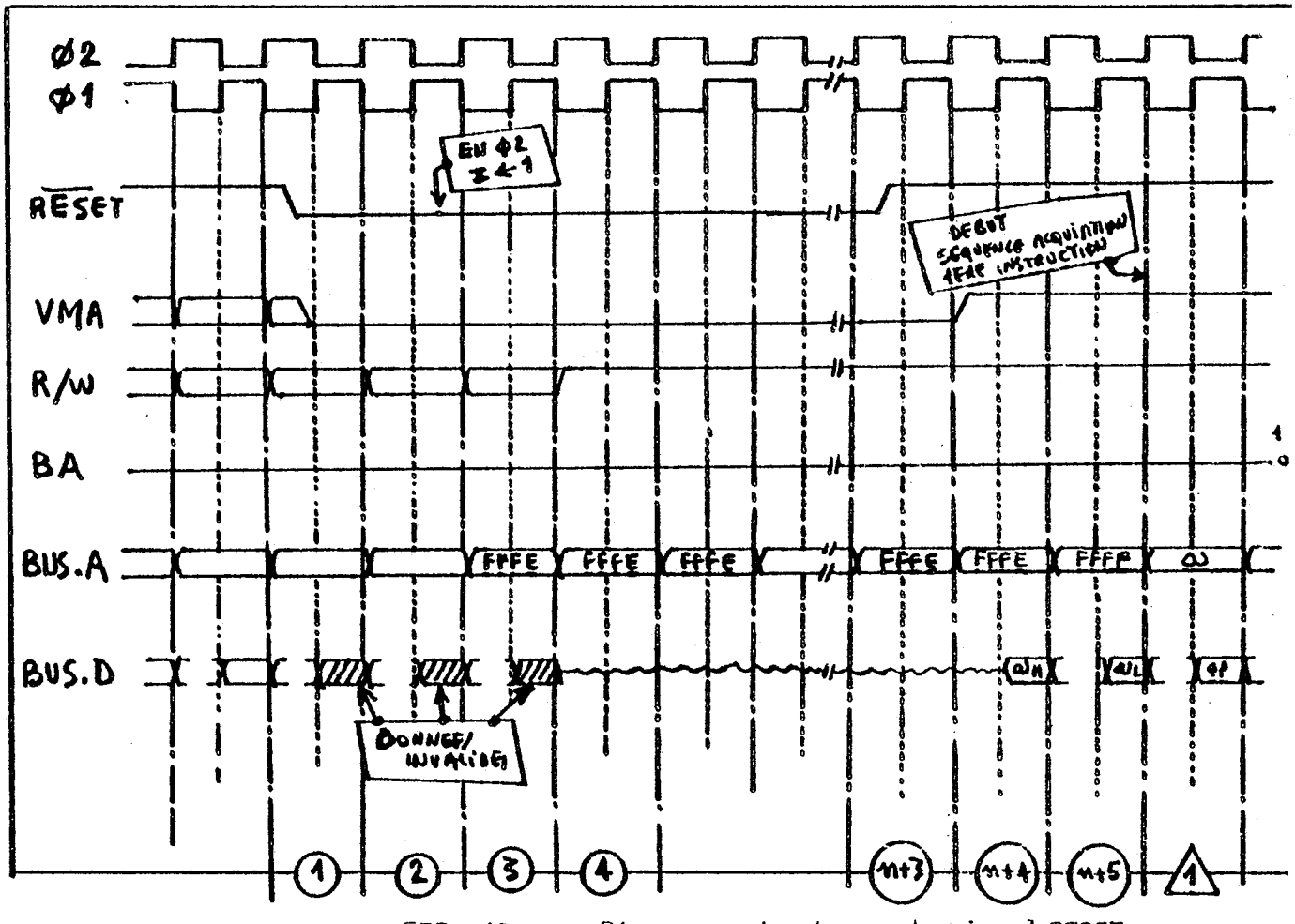


FIG. 49 - Diagramme des temps du signal RESET

La broche [$\overline{\text{RESET}}$] commande directement la mise à zéro de la broche VMA (avec un retard d'environ 70 ns). L'échantillonnage du signal externe RESET se fait pendant la phase Ø2 de chaque cycle.
La simulation électrique de la bascule RESET (Figure 18) nous a montré que si le signal [RESET=0] arrive 100 ns (au moins) avant le front descendant de la phase Ø2, l'initialisation interne démarre à partir de la phase Ø1 du cycle suivant, sinon l'initialisation interne du microprocesseur est retardée d'un cycle.

3.3. - Barrières d'échantillonnage du signal HALT

Le signal [$\overline{\text{HALT}}$] permet d'arrêter l'exécution des instructions par le microprocesseur. Ceci permet de réaliser l'exécution des programmes en mode "pas à pas" (*).

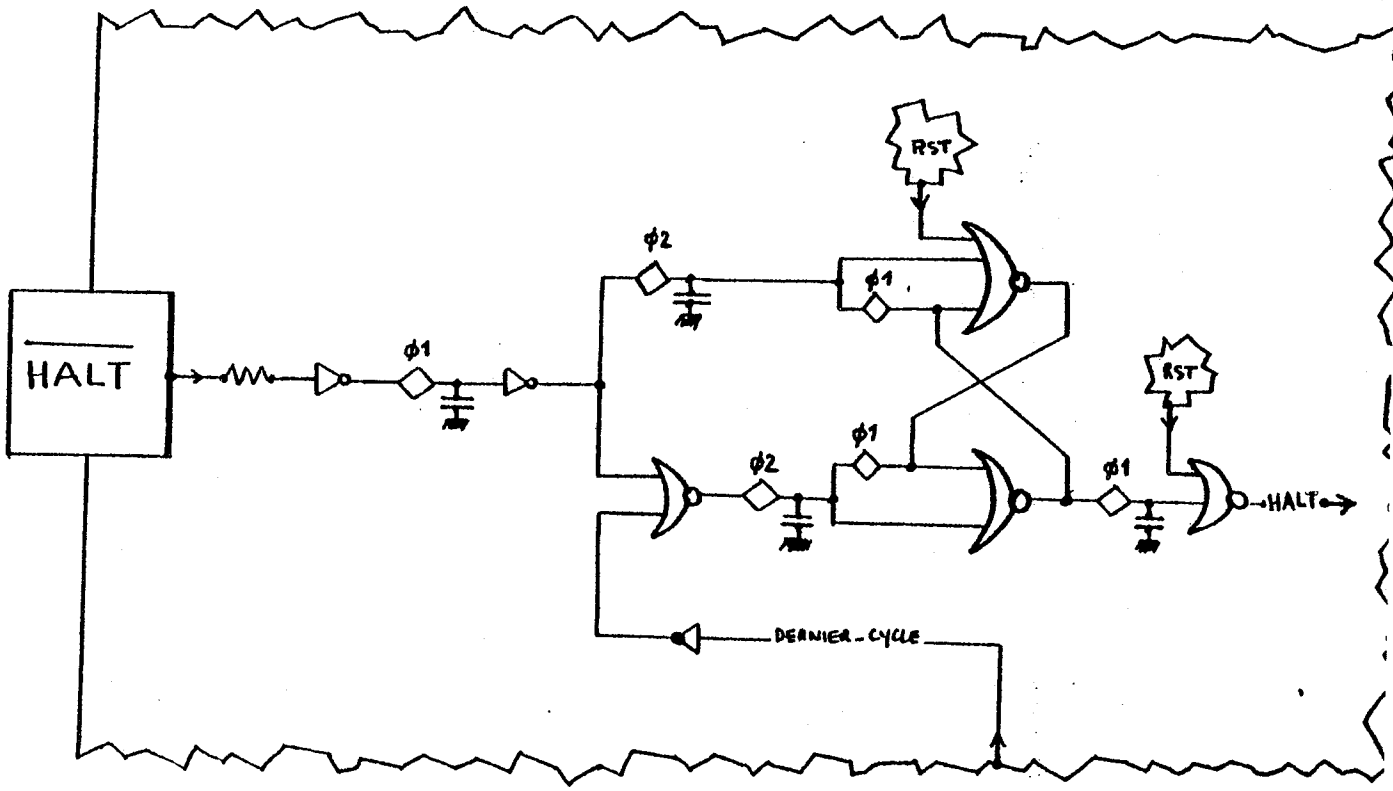


FIG. 50 - Bascule de mémorisation du signal HALT.

Le signal [$\overline{\text{HALT}}$] , comme sa notation l'indique, est actif au niveau bas.

Le séquenceur teste le signal $\overline{\text{HALT}}$ seulement pendant le dernier cycle de l'instruction en cours.

(*) Exécution d'une instruction suivie d'un arrêt.

En effet, le signal présent à l'entrée de la broche $\overline{HALT}$ est soumis à deux barrières d'échantillonnage. La première barrière est constituée par la phase Ø1, la deuxième est constituée par le signal DERNIER-CYCLE qui est généré par le bloc [ACQUISITION].

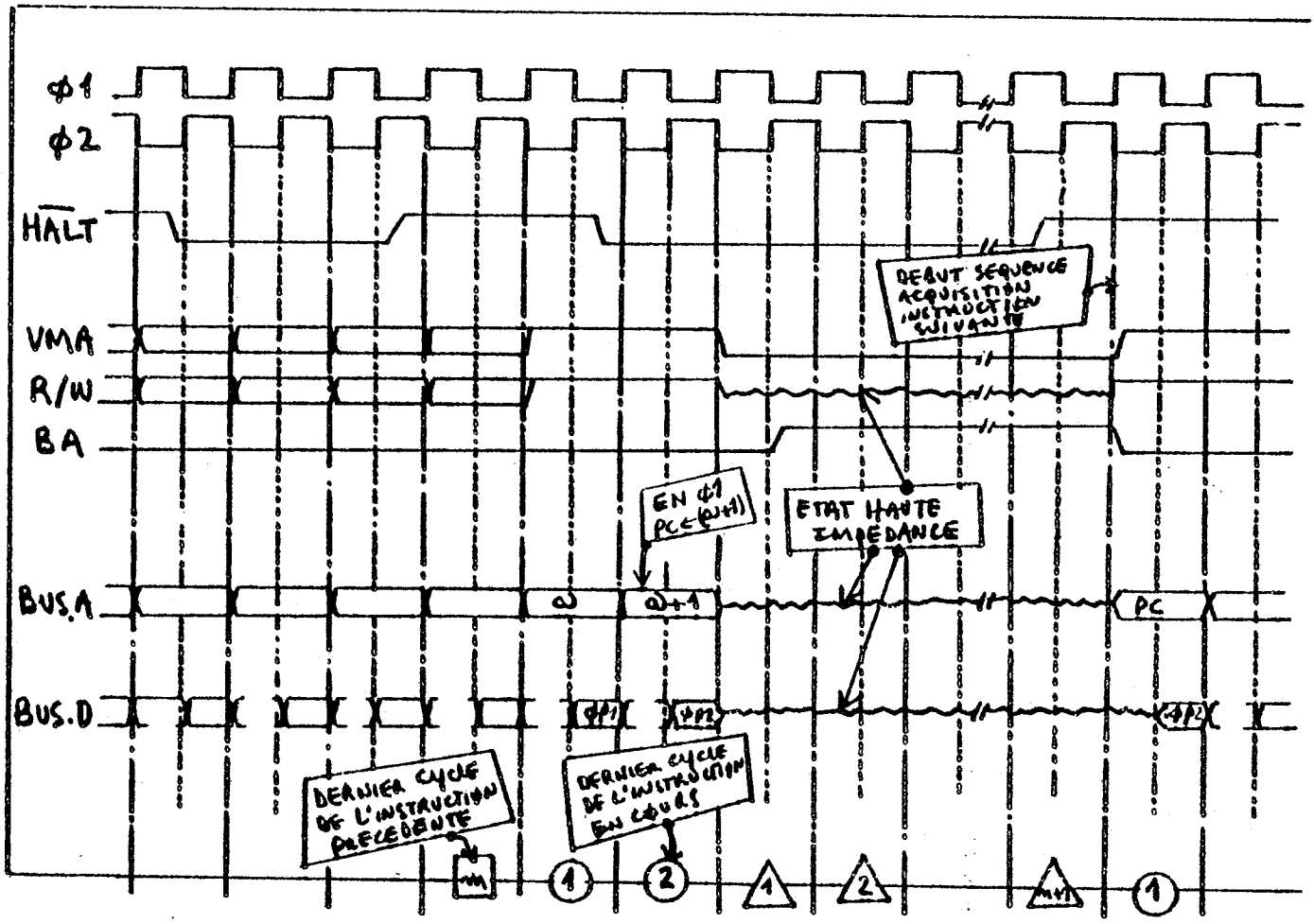


FIG. 51 - Diagramme des temps du signal HALT.

La prise en compte et le traitement du signal $\overline{HALT}$ dépendent de l'état d'ouverture de ces deux barrières. Pour être pris en compte, le signal [$\overline{HALT}$=0] doit être présent pendant la phase Ø1 du dernier cycle de l'instruction en cours. La remise à zéro de la bascule de mémorisation est commandée par la montée du signal HALT [$\overline{HALT}$=1] à l'entrée de la broche.

Le microprocesseur sort du mode HALT un cycle après la montée du signal [$\overline{HALT}$].

3.4. - Barrières d'échantillonnage du signal NMI

Le signal $\overline{NMI}$ permet d'interrompre l'exécution du programme en cours pour se brancher au programme du traitement des interruptions non masquables.

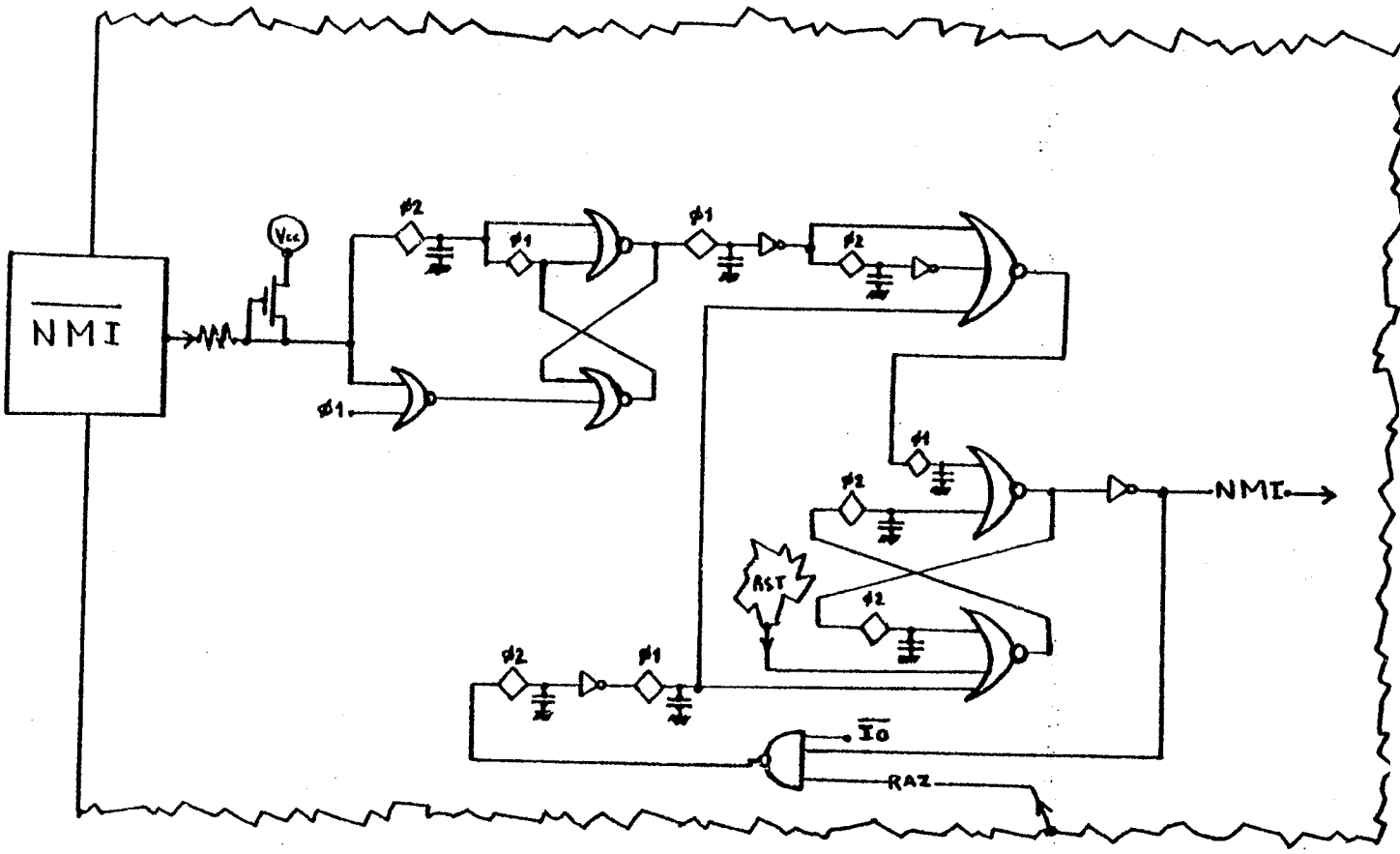


FIG. 52 - Bascule de mémorisation du signal NMI

L'entrée [$\overline{NMI}$] du microprocesseur possède un transistor de charge à déplétion (pull - up interne), aussi, tout circuit à drain-ouvert peut-il générer l'interruption non masquable NMI.

On distingue trois parties sur le schéma de la figure(52) :

- échantillonnage et détection du front descendant (*) du signal $\overline{NMI}$,
- bascule de mémorisation (type RS),
- échantillonnage du signal RAZ (signal de remise à zéro de la bascule) qui est généré par le bloc [OPERATION.II].

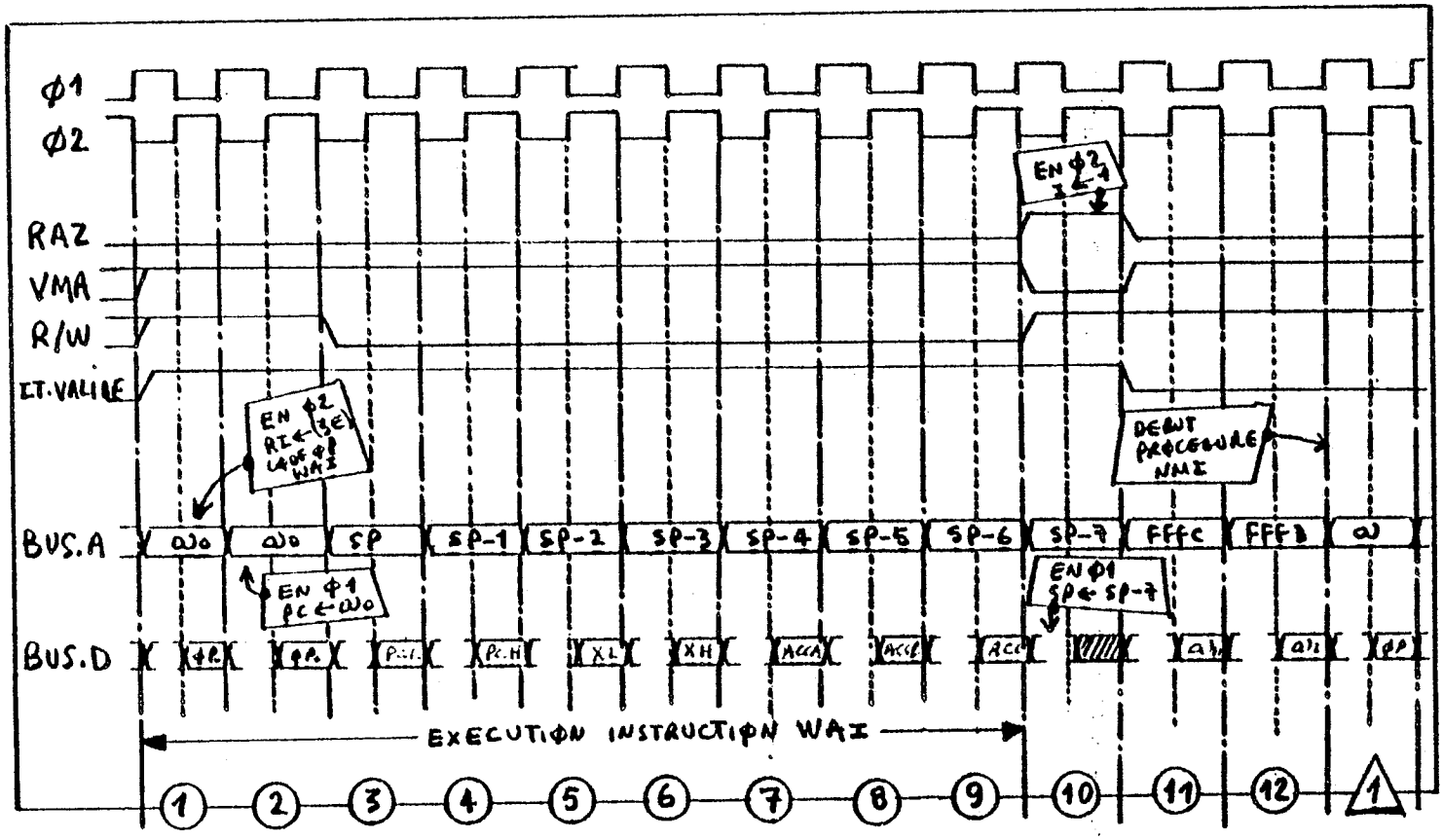


FIG. 53 - Diagramme des temps du signal NMI.

Le traitement de l'interruption NMI commence par l'exécution de l'instruction WAI, ensuite par un branchement au programme NMI.

(*) Passage du signal NMI de 1 à 0.

3.5. - Barrières d'échantillonnage du signal IRQ

Le signal $\overline{IRQ}$ permet d'interrompre l'exécution du programme en cours pour se brancher au programme du traitement des interruptions masquables.

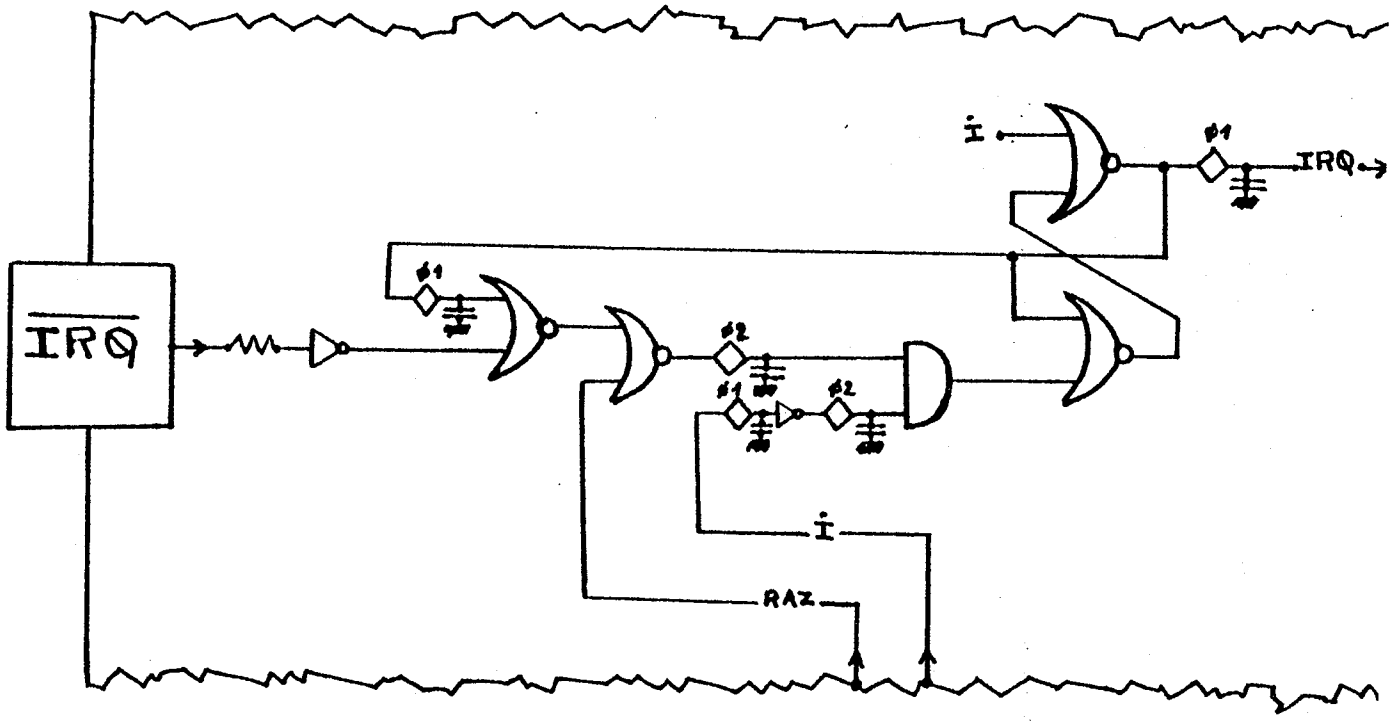


FIG. 54 - Bascule de mémorisation du signal IRQ.

Le signal [$\overline{IRQ}$], comme sa notation l'indique, est actif au niveau bas.

Comme pour le NMI, on distingue trois parties sur le schéma de la bascule de mémorisation du signal IRQ :

- échantillonnage et mise à 1 de la bascule IRQ,
- bascule de mémorisation (type RS),
- échantillonnage de la remise à zéro (RAZ, I).

Notons que si le masque I est positionné à 1, le signal [$\overline{IRQ}$=0] n'est pas pris en compte.
Le masque (I) sert de signal de remise à zéro de la bascule de mémorisation et constitue une barrière pour le signal [$\overline{IRQ}$=0].

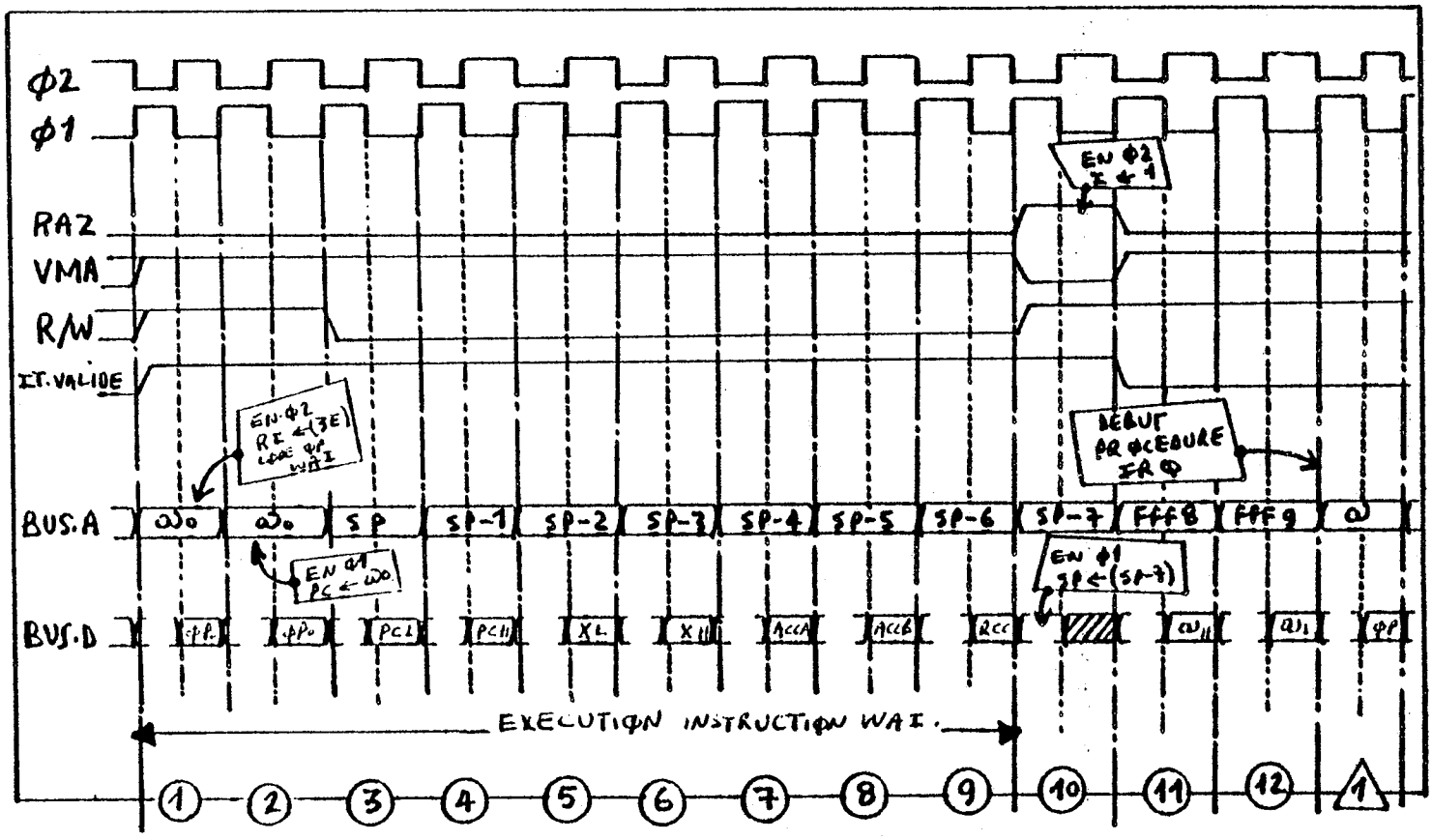


FIG. 55 - Diagramme des temps du signal IRQ.

Le traitement de l'interruption IRQ commence par l'exécution de l'instruction WAI, ensuite par un branchement au programme IRQ.

X - ANOMALIES DE FONCTIONNEMENT

1. - Au niveau du jeu d'instructions

1.1. - Codes invalides

Le contrôle assuré par le bloc [ACQUISITION] du séquenceur est commun à toutes les instructions. Le contrôle assuré par le bloc [ADRESSAGE] est, quant à lui, commun à l'ensemble des instructions appartenant à un même groupe (premier e etape d'interpretation).

Ceci est dû au codage horizontal du jeu d'instructions du 6800.

Le décodage du code opération, au niveau du bloc [ADRESSAGE] , se fait essentiellement sur les quatre bits de poids fort.

Donc, à ce stade de l'exécution, aucune distinction n'est faite entre un code opération valide et un code opération invalide.

Le traitement spécifique à une instruction (respectivement à un code invalide) ne s'effectue que pendant la séquence opération (deuxième etape d'interprétation).

Le codage des instructions a été défini de façon à permettre un maximum de souplesse et de facilité de décodage par la partie contrôle, notamment pour ce qui est du regroupement des actions communes à plusieurs instructions (codage vertical).

Dans le cas du chargement d'un code invalide, dans le registre instruction, l'exécution qui s'en suit, se déroule en deux étapes et ceci de manière différente.

Pendant les séquences ACQUISITION et ADRESSAGE, l'exécution est semblable à celle d'un code opération valide.

Ensuite, pendant la séquence OPERATION, l'exécution peut se poursuivre d'une manière différente de celle d'une instruction valide (selon le code invalide).

Cette différence se traduit au niveau de :

- aiguillage des signaux de contrôle dans le sequenceur,
- génération des commandes dans l'interface de commande.

En effet, cette différence d'exécution au niveau de l'interface de commande se caractérise par l'activation de certaines commandes qui se traduit dans la partie opérative par des actions élémentaires qui ne correspondent pas toujours à celles des instructions valides.
De même au niveau du séquenceur, l'aiguillage conditionnel des signaux à travers les blocs de contrôle n'est pas toujours semblable à celui déterminé par l'exécution des instructions valides.

Les deux paragraphes suivants, nous presentent le decodage de certains codes invalides dans la partie controle du 6800.

1.1.1. - Décodage au niveau du séquenceur

Dans le cas général, le contrôle du séquencement, durant l'exécution des codes invalides, est semblable à celui des instructions valides, exception faite pour certains codes invalides.

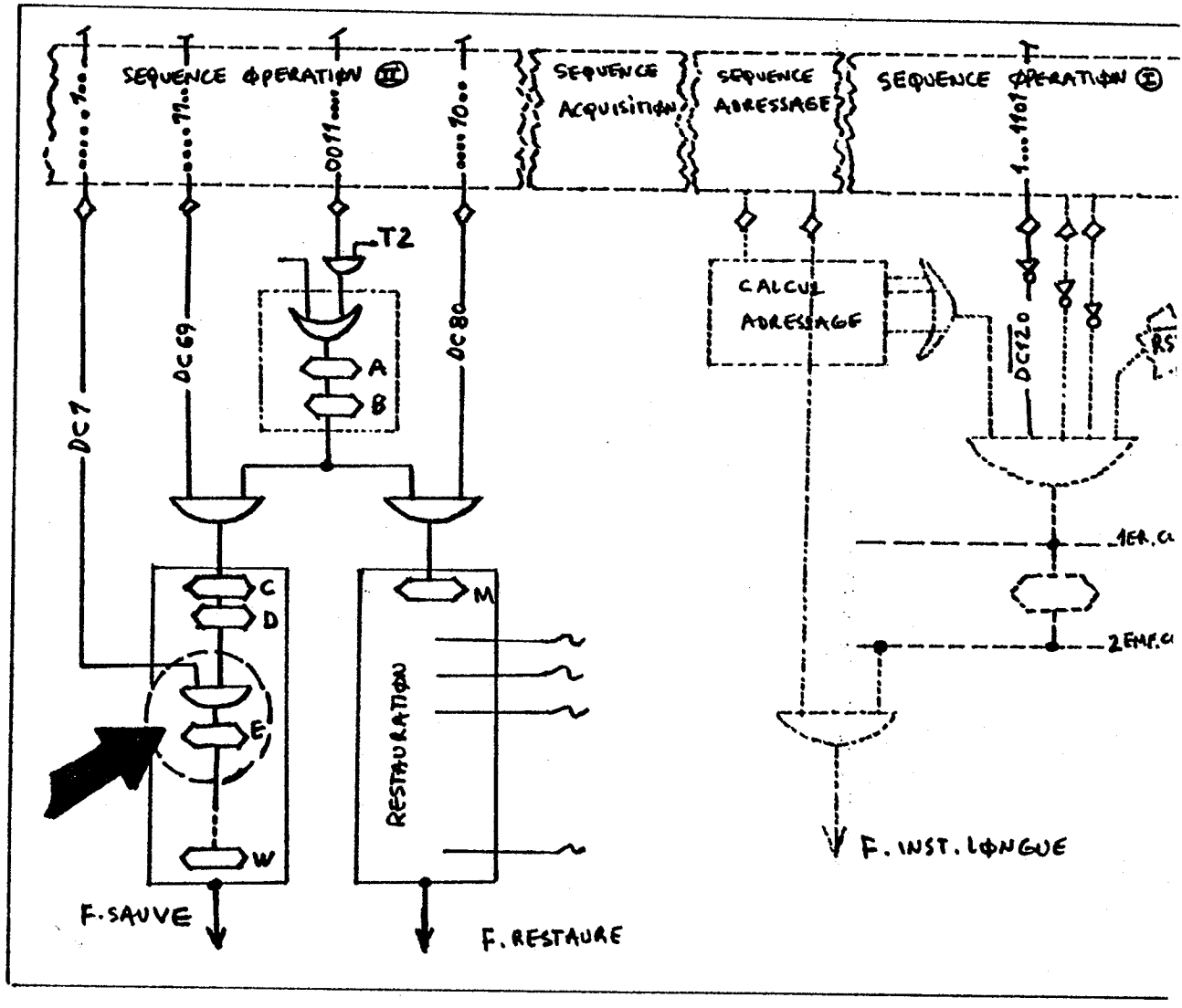


FIG. 56 - Décodage par le séquenceur des codes invalides (3C, 3D, 9D, DD)

L'exemple de la figure (56) nous montre le cas particulier du séquencement des codes invalides (3C, 3D, 9D, DD) par les blocs [OPERATION.II] et [OPERATION.I].

Le monome du décodeur DC 112 = ($\overline{I7}$. $\overline{I6}$. I5 . $\overline{I4}$) permet de reconnaître tous les codes appartenant au groupe d'instruction (V) et parmi eux, les codes 3C, 3D, 3E et 3F.

Les deux premiers codes sont des codes invalides, les codes 3E et 3F sont ceux des instructions WAI et SWI.

Pendant l'exécution de l'un de ces quatre codes, les monomes DC 112 et DC 69 = (I3 . I2) prennent la valeur logique '1' ce qui permet l'activation des étages A, B, C et D et ceci par propagations successives dans les registres à décalage correspondants.

L'activation de l'étage A a lieu au deuxième cycle (T_2) de la séquence ACQUISITION.

Le premier étage (M) du registre à décalage [RESTAURATION] n'est pas activé à partir de l'étage (B) car le monome DC 80 : ($I3.\overline{I2}$) prend la valeur zéro durant l'exécution de l'un quelconque de ces quatre codes.

Le monome DC1 = (I1), par contre, ne prend pas la même valeur qu'il s'agisse de l'exécution des codes (3C, 3D) ou de celle des codes (3E,3F).

En effet, pendant l'exécution des instructions WAI ou SWI, la condition DC1 prend la valeur logique '1', ce qui permet l'activation de l'étage (E) à partir de l'étage (D) par propagation dans le registre à décalage [SAUVEGARDE]. L'activation de l'étage (E) permet, par des propagations

successives dans le registre [SAUVEGARDE], le passage du contrôle au circuit [BOUCLE ATTENTE PAR WAI] qui à son tour permet le passage du contrôle au bloc [ACQUISITION].

Une nouvelle exécution démarre.

Pendant l'exécution des codes invalides (3C, 3D), la condition DC1 prend la valeur '0' ce qui empêche l'activation de l'étage (E) à partir de l'étage (D).

Ainsi, l'étage (W) n'est jamais activé pendant l'exécution de ces deux codes invalides. Aussi, le registre à décalage [SAUVEGARDE] ne permet plus le passage du contrôle au circuit[AUCUNE ATTENTE POUR WAI] qui à son tour se trouve dans l'impossibilité d'activer le bloc [ACQUISITION].

Nous aboutissons à un blocage dans le séquenceur, car aucun bloc de contrôle n'est actif.

Aucune nouvelle instruction n'est chargée. L'exécution qui se poursuit est décrite en détail en Annexe.2.

Le même phénomène arrive pendant l'exécution des codes invalides (9D, DD), car dans ce cas, le bloc [OPERATION.I]se trouve dans l'impossibilité de passer le contrôle au bloc [ACQUISITION] par le signal (F.INSTR.LONGUE). En effet, pendant l'exécution des codes invalides (9D, DD) le monome DCN 120 = ($\overline{I7}+\overline{I6}+\overline{I2}+I1+I0$) prend la valeur logique '0' ce qui empêche l'activation de l'étage (1ER.CYCLE.CALCUL) du bloc [OPERATION.I].

Ainsi, à partir du troisième cycle d'exécution, tous les registres à décalage du séquenceur se trouvent désactivés et un blocage, semblable au précédent, se produit.

1.1.2. - Décodage au niveau de l'interface de commandes

Nous considérons l'exemple de deux types de codes invalides dont l'exécution est tout à fait identique à celle d'une instruction valide pour l'un (même séquencement, même génération de commandes) alors qu'elle n'est que partiellement identique pour l'autre.

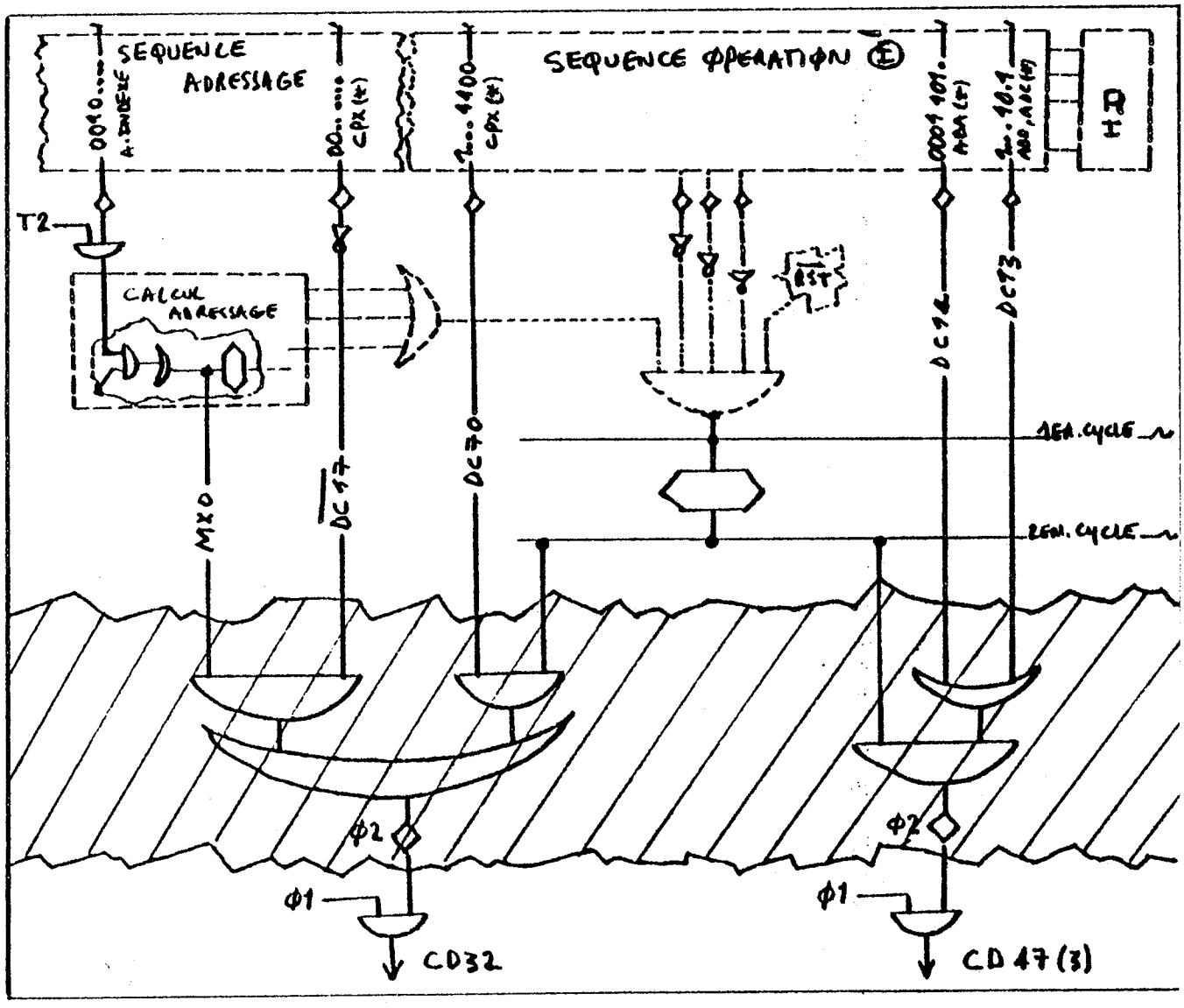


FIG. 57 - Décodage par l'interface de commandes des codes invalides (CC, DC, EC, FC).

Les monomes DCN 17 = (I7+I6) et DC 70 = (I7.I3.I2.$\overline{I1}$.$\overline{I0}$) permettent de rconnaître les codes opérations (FC,EC,DC,CC) et (BC,AC,9C,8C).

Notons que les quatre premiers codes, sont des codes invalides et les quatre derniers sont ceux de l'instruction CPX avec ses différents modes d'adressage.

L'exécution de l'un de ces huit codes positionne la condition DC 70 à '1' ce qui permet de générer la commande (CD 32 = 1) pendant le deuxième cycle de la séquence opération.

La condition DCN 17 ne permet la génération de la commande [CD 32], que pour les situer dans un groupe dont le mode d'adressage est indexé (cas des codes AC et EC).

Durant leur traitement, les codes AC et EC se comportent de la même manière vis-à-vis de la génération de la commande [CD 32].

Il en est de même pour la génération des autres commandes (à travers d'autres monomes) dans les différentes étapes de l'exécution de ces deux codes.

Ainsi, l'exécution du code invalide EC est identique à celle de l'instruction CPX(AC) en mode d'adressage indexé. Il en est de même de l'exécution des codes invalides FC, DC, CC avec celle de l'instruction CPX pour les modes d'adressage respectifs étendu, direct et immédiat.

La modification des expressions booléennes de certains monomes du décodeur permet de changer le traitement des codes invalides tout en laissant intact le traitement des codes valides.

Par exemple, pour supprimer l'identité d'exécution entre les quatre codes invalides (CC, DC, EC, FC) et l'instruction CPX, il suffit de modifier la condition DC 70 comme suit :

$$DC\ 70 = (I7.\overline{I6}.I3.I3.I2.\overline{I1}.\overline{I0})$$

Avec cette nouvelle condition, la commande CD 32 est générée correctement pendant le traitement de l'instruction CPX, mais elle ne l'est plus lors de l'exécution de l'un des quatre codes invalides précédents, car dans ce cas on a : (DC 70 = 0).

L'exécution de ces quatre codes invalides et celle de l'instruction CPX ne sont plus que partiellement identiques, ce qui est le cas réel de certains codes invalides avec des instructions du 6800.

Le deuxième exemple de la Figure(57) nous montre que le monome $DC\ 14 = (\overline{I7}.\overline{I6}.\overline{I5}.I4.I3.\overline{I2}.I1)$ permet la génération de la commande [CD 47(3)] respectivement pour l'instruction ABA($1B$) et le code invalide (1A), pendant le deuxième cycle de la séquence opération.

L'exécution du code invalide (1A) est totalement identique à celle de l'instruction ABA.

Par contre, l'exécution du code invalide (1D) n'est que partiellement identique à celle de l'instruction ABA.

En effet, pendant l'exécution du code invalide (1D), la condition DC.14 prend la valeur logique '0' ce qui empêche la génération de la commande [CD.47(3)] pendant le cycle [2EME.CYCLE.CALCUL].

1.2. - Traitement des instructions opérant sur 16 bits

Le bloc [ADRESSAGE] permet la validation de toutes les commandes nécessaires au chargement de l'opérande de l'instruction en cours.

La lecture de la donnée qui détermine l'adresse de l'opérande pour une instruction, dont le mode d'adressage est étendu, s'effectue en deux étapes et dans l'ordre suivant :
- lecture de l'octet de poids fort de l'adresse (@15,...,@8),
- lecture de l'octet de poids faible de l'adresse (@7,...,@0).

Cette technique de lecture est également utilisée pour le chargement des opérandes de type long (sur deux octets).

Dans le cas général, le traitement de la séquence opération pour des instructions qui nécessitent des opérandes longs (deux octets) ne pose aucun problème, exception faite pour l'instruction CPX.

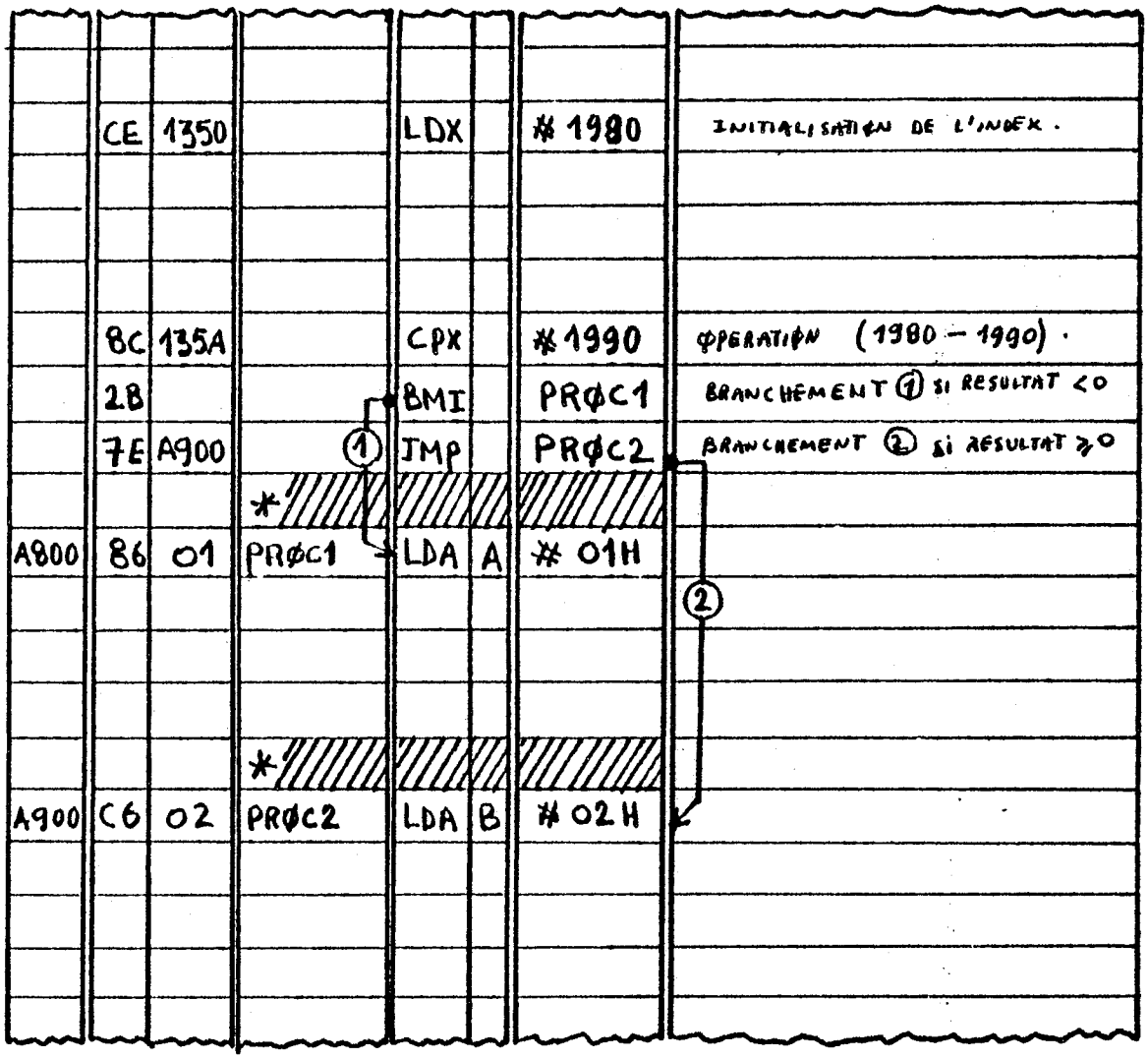
L'aspect performance en temps d'exécution, étant primordial, le traitement de l'opération de comparaison des octets de poids fort (XH-MH) se déroule pendant le chargement de l'octet de poids faible de l'opérande. L'opération de comparaison des octets de poids faible est ensuite réalisée.

Cette façon de réaliser l'exécution de l'instruction CPX ne permet pas de positionner correctement l'indicateur de signe (N) du registre d'état RCC.

Le positionnement correct de (N) dépend aussi bien de l'opération de comparaison des octets de poids fort que de celles des octets de poids faible, ce qui n'est pas fait pour l'instruction CPX.

Exemple :

Soit à exécuter le programme suivant par le microprocesseur 6800 :



Au moment de l'exécution de l'instruction CPX, le contenu du registre index est (X = 1980) (*).

L'opération de comparaison entre l'index et l'opérande de l'instruction CPX, devrait donner un résultat négatif (1980 - 1990 = -10) (*).

(*) Valeurs décimales.

Normalement, après l'exécution de l'instruction CPX, le branchement dû à l'instruction BMI devrait s'effectuer au début de la procédure PROC1. En réalité, l'exécution de l'instruction CPX dans cet exemple, positionne l'indicateur de signe à zéro (N=0) ce qui signifie que le résultat de la comparaison est positif ou nul. Aussi, après l'exécution des instructions BMI et JMP, le branchement a bien lieu au début de la procédure PROC2, ce qui est l'inverse de ce que l'on veut obtenir.

Ceci est dû au fait que, lors de l'exécution de CPX, le positionnement de l'indicateur de signe (N) se fait uniquement en fonction de l'opération de comparaison des octets de poids fort.

Notons qu'il n'est pas facile de remédier, d'une manière simple et souple, à cette anomalie de fonctionnement.

Deux solutions sont envisageables pour supprimer cette anomalie.

La première est coûteuse en matériel et en temps de conception car elle consiste à redessiner les masques de toute la partie contrôle, avec au bout de l'épreuve une augmentation non négligeable de la surface du circuit intégré.

La deuxième est nettement moins coûteuse en matériel, mais elle créée des contraintes non négligeables au niveau de l'écriture des programmes. Cette solution consiste à changer le format des instructions qui manipulent des données de 16 bits et à inverser l'ordre de chargement des octets de poids fort et de poids faible.

Le format de ces instructions devient : [COP] [ML] [MH] (*).

(*) : COP est le code opération de l'instruction
ML est l'octet de poids faible de la donnée
MH est l'octet de poids fort de la donnée.

Ces conventions d'écriture sont utilisées dans le jeu d'instruction de certains microprocesseurs, tels que : Z 80, I 8080, I 8085 ...

Il est indéniable que ce type de format d'instruction peut être très génant pour l'utilisateur, surtout quand la donnée à manipuler est une adresse.

2. - Au niveau du système d'interruptions

2.1. - Action des signaux d'interruption [NMI, IRQ].

Le choix des barrières de protection du 6800 contre la prise en compte des interruptions [NMI, IRQ], lors d'une séquence d'initialisation, n'est pas efficace. Cette défaillance de protection peut être à l'origine de graves anomalies de fonctionnement.

Pendant la séquence d'initialisation [$\overline{\text{RESET}}$=0] , les bascules de mémorisation des interruptions [NMI, IRQ] sont remises à zéro deux cycles après le passage à zéro du signal [$\overline{\text{RESET}}$].

Le masque d'interruption de l'interruption masquable [IRQ] est également positionné (I=1).

La remise à zéro des bascules d'interruption et le positionnement du masque de l'IRQ, pensant une séquence d'initialisation, ne constituent pas une barrière efficace pour éviter la prise en compte des IT.

Tant que le signal externe (RESET=0) est present, la remise à zero des bascules d'interruptions et le positionnement du masque (I) de l'IRQ se font periodiquement pendant tous les cycles.
A la montée du signal (RESET), les signaux d'interruptions,qui arrivent, peuvent etre memorisés et traités avant l'execution de la procedure d'initialisation du systeme.

Les conditions qui entraînent la prise en compte des interruptions dépendent du type du signal d'IT :

- si le signal [$\overline{\text{NMI}}$=0] arrive à la montée du signal [$\overline{\text{RESET}}$] , alors il est mémorisé et son traitement se fait avant l'exécution de la procédure d'initialisation.

Dans certaines conditions (*), si le signal [$\overline{IRQ=0}$] est présent, à la montée du signal [$\overline{RESET}$], alors il est mémorisé et son traitement se fait avant l'exécution de la procédure d'initialisation du système.

A la mise sous tension du microprocesseur et avant l'arrivée du signal [$\overline{RESET=0}$] , la prise en compte et le traitement des signaux d'interruption [NMI,IRQ] dépend de l'exécution en cours.

Deux cas peuvent se présenter, selon le contenu du registre instruction :

premier cas :

Le contenu du registre instructions est l'un des codes invalides (3C, 3D, 9D, DD) . Ceci peut arriver, soit pendant la stabilisation de l'alimentation, soit pendant une opération de chargement de l'instruction en cours. Les signaux d'interruption [$\overline{HALT}$,$\overline{NMI}$,$\overline{IRQ}$] ne sont pas traités par le microprocesseur.

deuxième cas :

Le contenu du registre instruction est différent des codes invalides (3C, 3D, 9D, DD). Les signaux d'interruption [$\overline{HALT}$,$\overline{NMI}$] sont pris en compte et traités normalement. La prise en compte est le traitement du signal [$\overline{IRQ=0}$] dépend évidemment du masque (I).

(*) Voir ANOMALIE 2.

2.2. - Prise en compte de l'interruption [NMI] lors d'un RESET

Une interruption non masquable (NMI) qui arrive pendant la séquence d'initialisation interne [$\overline{\text{RESET}}$=0] n'est pas mémorisée car la remise à zéro de la bascule (NMI) se fait pendant tous les cycles.

De plus, la mémorisation du signal [$\overline{\text{NMI}}$=0] se fait sur le front descendant (contrairement à celle du signal [IRQ] qui se fait sur niveau).

Cependant, si l'interruption NMI arrive à la montée du signal [$\overline{\text{RESET}}$] (à partir du cycle qui suit la montée du $\overline{\text{RESET}}$), alors elle est mémorisée et son traitement peut se faire avant l'exécution de la procédure d'initialisation du système (Figure 58).

Or, la priorité du signal [$\overline{\text{RESET}}$] étant supérieure à celle du [$\overline{\text{NMI}}$], par conséquent le traitement du signal [$\overline{\text{RESET}}$] (initialisation interne et accés à la procédure (*) d'initialisation du système) doit se faire avant celui du [$\overline{\text{NMI}}$] (sauvegarde des registres internes dans la pile et accés à la procédure du NMI).

Tout ceci constitue l'anomalie 1 dont les conséquences peuvent être très graves.

En effet, le traitement du NMI commence par la sauvegarde du contexte (registre PC,X,...) dans la zone mémoire adressée par le pointeur de pile SP.

Comme SP n'a pas été initialisé, la sauvegarde peut se faire dans n'importe quelle partie de l'espace mémoire, même à des adresses non autorisées.
De plus, le programme du NMI peut avoir des exécutions dangereuses :
- opération d'entrées/sorties,
- envoi de messages erronés dans le cas des systèmes multiprocesseurs.

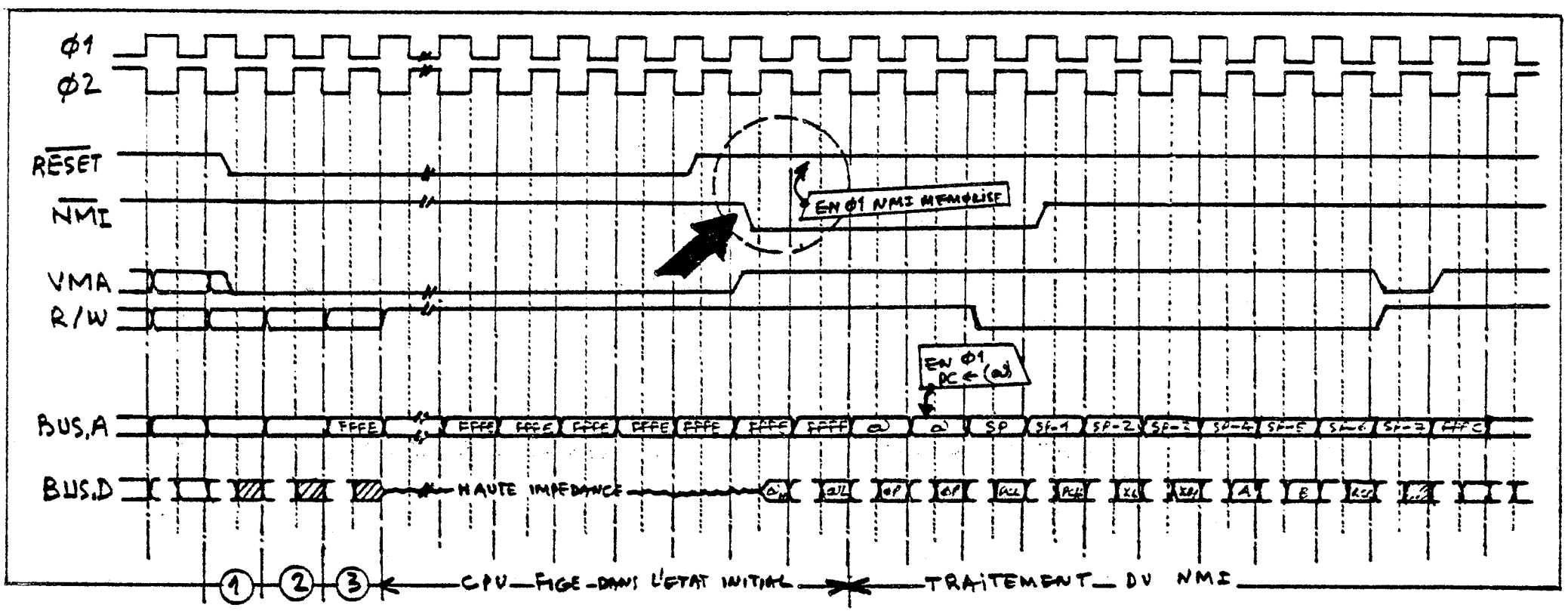
Ce n'est qu'apres le traitement du NMI que l'execution de la procedure d'initialisation pourrait avoir lieu.

(*) Execution d'au moins une instruction de cette procedure.

L'anomalie 1 a lieu si le signal [$\overline{NMI}$=0] arrive pendant les deux cycles qui suivent celui de la montée du signal [$\overline{RESET}$] .

FIG.58:

DIAGRAMME DES TEMPS POUR L'ANOMALIE.I . .185.



2.3. - Démasquage accidentel de l'interruption IRQ.

Une interruption masquable [IRQ] qui arrive pendant la séquence d'initialisation interne [$\overline{\text{RESET}}$=0] n'est pas mémorisée car le masque (I) est positionné à '1', la bascule de mémorisation de l'IRQ est remise à zéro pendant tous les cycles.

Si le signal [$\overline{\text{RESET}}$=0] arrive pendant l'exécution de l'instruction CLI, alors cette exécution s'arrête et le code opération (OE) reste dans le registre instruction durant toute la séquence d'initialisation interne.

Le registre instruction n'est pas modifié par la séquence d'initialisation interne. A la montée sur signal [$\overline{\text{RESET}}$], le masque (I) est donc remis à zéro car l'instruction CLI contenue dans le registre instruction est de nouveau exécutée.

En effet, l'exécution de la séquence opération de l'instruction CLI est contrôlée directement par les monomes du décodeur sans aucune validation des commandes par le bloc [OPERATION.I] comme elle devrait l'être.

Il en est de même pour l'exécution des instructions CLV, CLC, SEI, SEV, SEC, mais les conséquences de celles-ci ne sont pas aussi graves que celle de CLI.

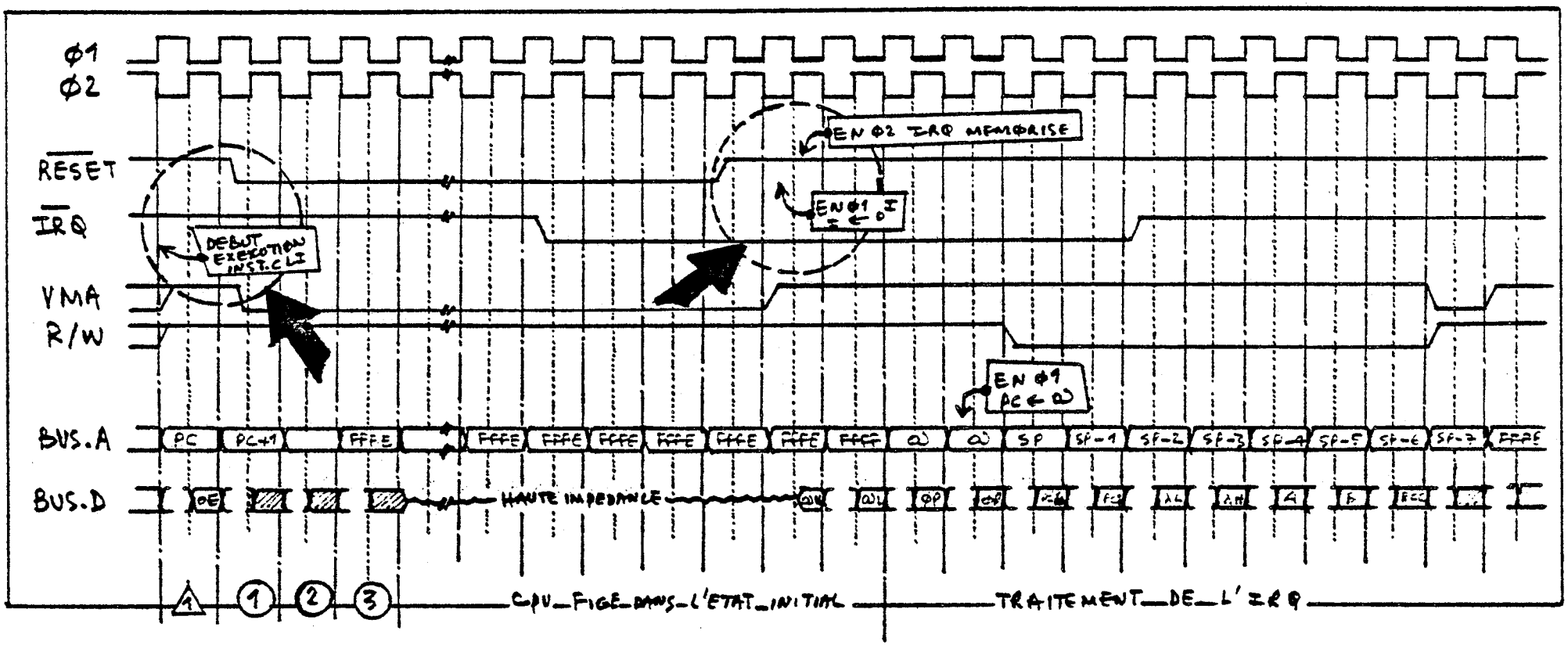
L'interruption IRQ est prise en compte, lors d'une séquence d'initialisation, si les deux conditions suivantes sont réalisées :
- le signal [$\overline{\text{RESET}}$=0] arrive pendant l'exécution de l'instruction CLI,
- le signal [$\overline{\text{IRQ}}$=0] est présent à la montée du signal [$\overline{\text{RESET}}$].

Comme pour le NMI (anomalie.1), après la mémorisation du signal [$\overline{\text{IRQ}}$=0] le traitement de celui-ci se fait avant l'exécution de la procédure d'initialisation du système. Ceci constitue l'anomalie 2, dont les conséquences sont semblables à celles de l'anomalie.1.

L'anomalie.2 a lieu si les deux conditions suivantes sont réalisées :

- le signal [$\overline{\text{RESET}}$=0] arrive pendant l'exécution de l'instruction CLI,
- le signal [$\overline{\text{IRQ}}$=0] est présent à la montée du signal $\overline{\text{RESET}}$.

DIAGRAMME DES TEMPS POUR L'ANOMALIE.II./FIG.59



3. - Résumé des anomalies des microprocesseurs 6800

Le microprocesseur 6800 (ainsi que les circuits de sa famille: 6802, 6801, ...) a subi plusieurs modifications depuis son apparition. Les plus importantes sont :

- le changement du processus de fabrication,
- le changement des masques du circuit intégré.

On distingue essentiellement deux versions :

a/ la première version, antérieure à 1977, est réalisée avec des Transistors de charge a enrichissement.

b/ la version actuelle, postérieure à 1977, quant à elle réalisée avec des Transistors de charge a depletion.

Les anomalies de fonctionnement du 6800 peuvent avoir différentes origines, pour les deux versions :

- erreur de conception,
- recherche de facilité de câblage de la partie contrôle.

Anomalie 1 : prise en compte de l'interruption NMI lors d'un RESET

Si le signal [$\overline{NMI}$=0] arrive pendant la montée du signal [$\overline{RESET}$] , l'interruption NMI est prise en compte par le microprocesseur et son traitement se fait avant celui de la procédure d'initialisation du système :

- elle existe sur les premières versions,
- elle existe sur les versions actuelles.

Raison apparente:

(ERREUR DE CONCEPTION).

Anomalie 2 : prise en compte de l'interruption IRQ lors d'un RESET

Si le signal d'initialisation [$\overline{RESET}$=0] arrive pendant l'exécution de l'instruction CLI, alors l'exécution de celle-ci est arrêtée, mais reprend à la montée du signal [$\overline{RESET}$] . La nouvelle exécution de l'instruction CLI, à la montée du signal [$\overline{RESET}$] , met le masque (I) à zéro. Si pendant ce temps une interruption IRQ est sollicitée, elle est prise en compte et son traitement se fait avant celui de la procédure d'initialisation du système :

- elle existe sur les premières versions,
- elle existe sur les versions actuelles.

Raison apparente:

(TROP GRANDE RECHERCHE DE FACILITE DE CABLAGE).

Anomalie 3 : mauvais positionnement des bits (N,V) par l'instruction CPX

L'instruction CPX réalise l'opération de comparaison sur 16 bits en deux étapes: on a d'abord la comparaison des octets de poids fort, ensuite la comparaison des octets de poids faible. Les bits (N) et (V) du registre des codes de condition RCC sont positionnés seulement en fonction de la comparaison des octets de poids fort.

- elle existe sur les premières versions,
- elle existe sur les versions actuelles.

Raison apparente:

(ECONOMIE SUR LE CABLAGE DE LA PARTIE CONTROLE).

Anomalie 4 : conflit entre les interruptions SWI, NMI

Si pendant l'exécution de l'instruction SWI, une interruption matérielle NMI arrive avant le dixième cycle du traitement de SWI, alors à la fin de la sauvegarde du contexte, les vecteurs d'adresse générés sont ceux de l'IRQ (FFF8-FFF9).
A la suite de ce conflit, on a un branchement vers la procédure de l'interruption IRQ (la moins prioritaire).

- elle existe sur les premières versions,
- elle n'existe pas sur les versions actuelles.

Raison apparente:

(ERREUR DE CONCEPTION).

Anomalie 5 : traitement de l'IT IRQ pendant l'exécution de CLI

- Sur les premières versions,

Si pendant l'exécution de l'instruction CLI, une interruption IRQ arrive, sa prise en compte (mémorisation et traitement) dépend de l'instruction (A) qui a précédé CLI :

1/ si le bit de poids le plus faible de l'instruction (A) est 0, alors l'interruption IRQ est mémorisée et traitée après l'instruction CLI.

2/ si le bit de poids le plus faible de (A) est 1, alors l'interruption IRQ est mémorisée et traitée après l'instruction qui suit celle de CLI.

- Sur les versions actuelles,

Il n'y a pas de confusion, l'interruption IRQ est mémorisée et traitée après l'instruction qui suit celle de CLI.

Raison apparente:
(ERREUR DE CONCEPTION).

Anomalie 6 : écriture intempestive par l'instruction TST

L'exécution de l'instruction TST entraine une réécriture de l'opérande en mémoire (ce qui n'est pas nécessaire). Ceci se passe pendant le dernier cycle de l'exécution par le positionnement des signaux suivants: VMA=1, R/W=0.

- elle existe sur les premières versions,
- elle n'existe pas sur les versions actuelles.

Raison apparente:
(ERREUR DE CONCEPTION).

XI - EVALUATIONS CONCRETES SUR LA REALISATION DES MASQUES DU 6800S

L'étude approfondie de l'architecture interne des microprocesseurs 6800 nous a montré l'existence de certaines anomalies de fonctionnement.

Ces anomalies de fonctionnement, sur l'ensemble des versions, ont des causes différentes :

- erreurs de conception (bascule NMI),
- trop grande recherche de facilité dans le câblage de la partie contrôle (démasquage accidentel de l'IRQ, exécution variée des codes invalides), ...

Le projet 6800S consiste à corriger tous les défauts de la version actuelle du 6800 par une modification partielle des masques du circuit intégré.

Les trois anomalies qui existent toujours y sont corrigées et les codes invalides detectés par l'integration d'un mecanisme de detection de pannes par codes invalides.
La compatibilite avec le 6800 reste totale, tant du point de vue du logiciel que de celui du brochage.

1. - Correction des anomalies du 6800 au niveau interne

1.1. - Correction de l'anomalie 1

L'anomalie 1 est due à une défaillance de protection de la prise en compte de l'interruption NMI lors d'une séquence d'initialisation.

En effet, sur la version actuelle :
Si le signal [$\overline{NMI}$=0] arrive pendant la montée du signal [RESET], l'interruption NMI est prise en compte par le microprocesseur et son traitement se fait avant celui de la procédure d'initialisation du système (peut entraîner de graves conséquences).

La suppression de cette anomalie de fonctionnement consiste à renforcer la barrière de protection de la prise en compte du signal [$\overline{NMI}$=0] en établissant un masque pendant la séquence d'initialisation.

Ce résultat peut être obtenu en procédant à une modification partielle de la bascule de mémorisation du NMI, en empêchant la mémorisation du signal [$\overline{NMI}$=0] lors de la montée du signal [RESET] (Figure 60)

Cette solution assure l'exécution d'au moins une instruction de la procédure d'initialisation du système. Ainsi, la première instructions de la procédure d'initialisation peut être utilisée selon les besoins :
- initialisation du pointeur de pile,
- démasquage externe du NMI par le matériel (systèmes utilisant un masque externe d'interruption: voir Annexe.2.
- masquage du NMI par le logiciel (systèmes n'utilisant pas de masque externe d'interruption par le matériel).

La solution du masquage du NMI par le logiciel se fait comme suit :

La première instruction est utilisée pour positionner un octet (MI) de la mémoire. Cet octet permet d'indiquer si la séquence d'initialisation du système est terminée ou non (modifié à la fin de cette procédure).

Si une interruption NMI arrive pendant (ou après) l'exécution de la première instruction, elle est mémorisée et son traitement se fait à la fin de l'instruction en cours.

La procédure du NMI doit commencer par tester le contenu de l'octet (MI) et, suivant le résultat du test, soit l'exécution de la procédure continue (initialisation terminée), soit l'exécution de celle-ci s'arrête et un branchement vers la procédure d'initialisation a lieu (initialisation non terminée).

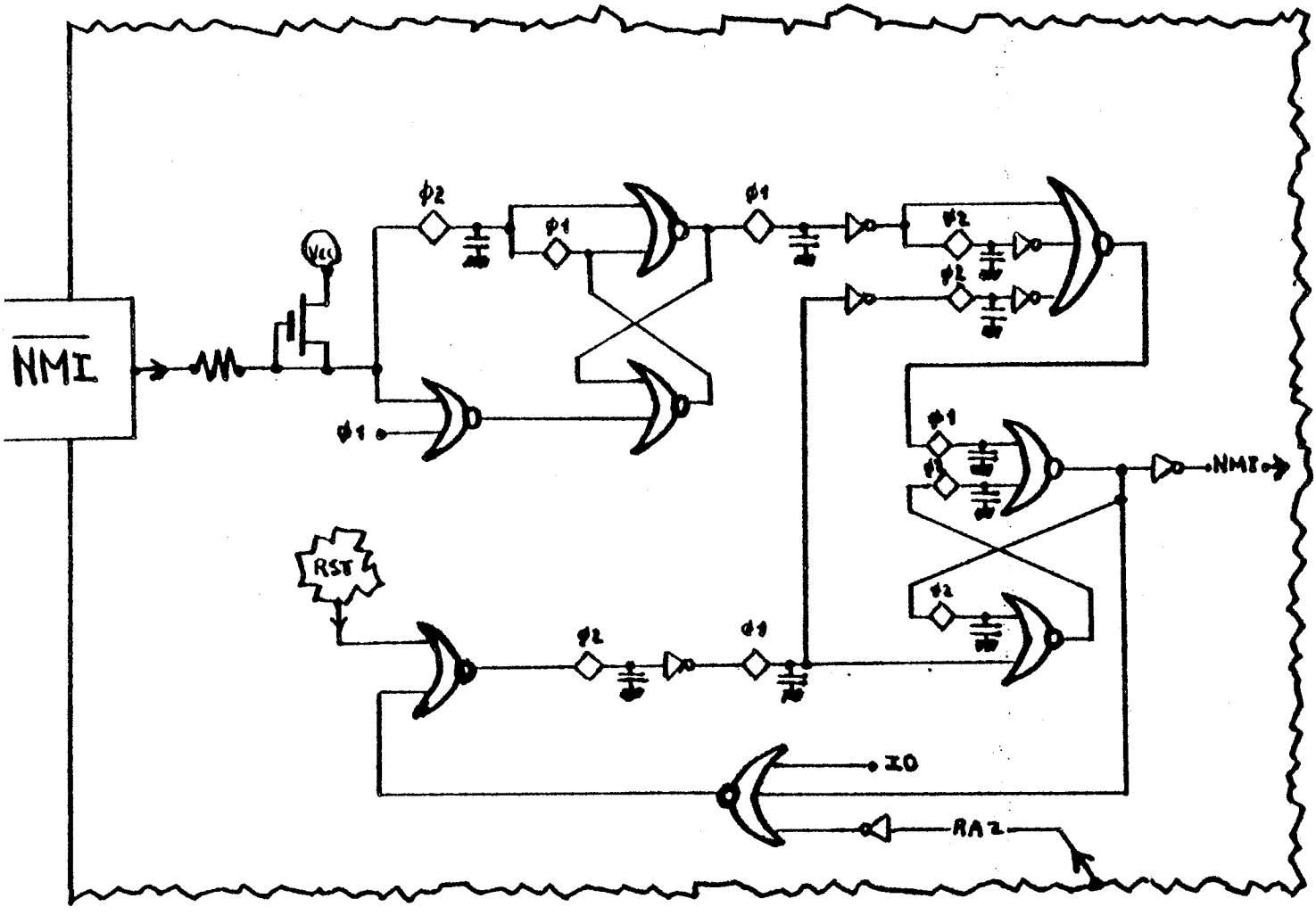


FIG. 60 - Bascule de mémorisation de l'IT [$\overline{NMI}$] / 6800S.

Note:
$\overline{RST}$ ...signal d'initialisation
RAZ(GS8) ...remise a zero des IT
I0 ...bit zero de l'instruction en cours, permet d'viter la remise a zero de la bascule du NMI par l'instruction SWI.

La nouvelle bascule de mémorisation des interruptions non masquables permet de masquer la prise en compte des signaux [$\overline{NMI}$] pendant les deux cycles qui suivent la montée du signal [$\overline{RESET}$] :

Ceci est réalisé par l'introduction d'un registre à décalage (à deux étages) ce qui permet d'établir un masque interne (dynamique) pour les interruptions NMI.

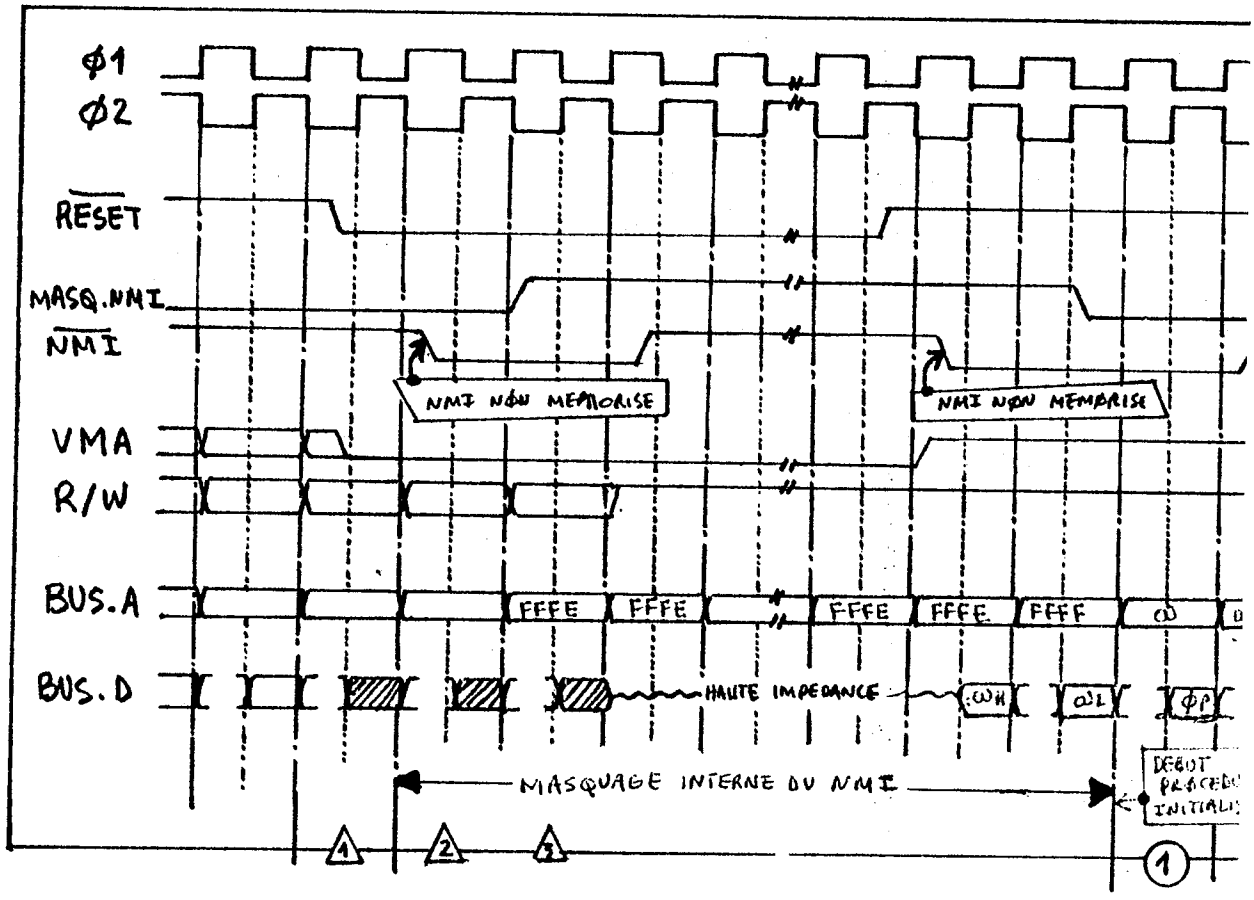


FIG. 61 - Diagramme des temps pour la suppression de l'anomalie 1.

(*) MASQUE NMI : masque interne de l'interruption NMI.

1.2. - Correction de l'anomalie 2

L'anomalie 2 est due à un mauvais câblage de la partie contrôle concernant le traitement du code opération de l'instruction CLI (de même pour les instructions: CLV, CLC, SEI, SEV, SEC).

La séquence opération de toutes ces instructions n'est pas contrôlée par le bloc (OPERATION.I) comme elle devrait l'être
(trop grande recherche de facilité dans le câblage).

En effet, sur la version actuelle,
Si le signal $\overline{\text{RESET}}$=0 arrive pendant l'exécution de l'instruction CLI, l'exécution de celle-ci est arrêtée, mais reprend à la montée du signal RESET.

La nouvelle exécution de l'instruction CLI, à la montée du signal RESET , met le masque (I) à zéro. Si pendant ce temps une interruption IRQ est sollicitée, elle est prise en compte par le microprocesseur et son traitement se fait avant celui de la procédure d'initialisation du système

- (peut entraîner de graves conséquences).

La suppression de cette anomalie consiste à rétablir le contrôle, par le séquenceur, de l'exécution des instructions CLI, CLV, CLC, SEI, SEV, SEC. Cette solution assure le "NON DEMASQUAGE" accidentel de l'interruption IRQ lors de la montée du signal RESET.

Le masque (I) reste positionné à '1' quelque soient les conditions d'exécution qui ont précédé la séquence d'initialisation.

L'utilisateur reste tout à fait libre du choix du démasquage ou non de l'IRQ pendant le traitement de la procédure d'initialisation du système.

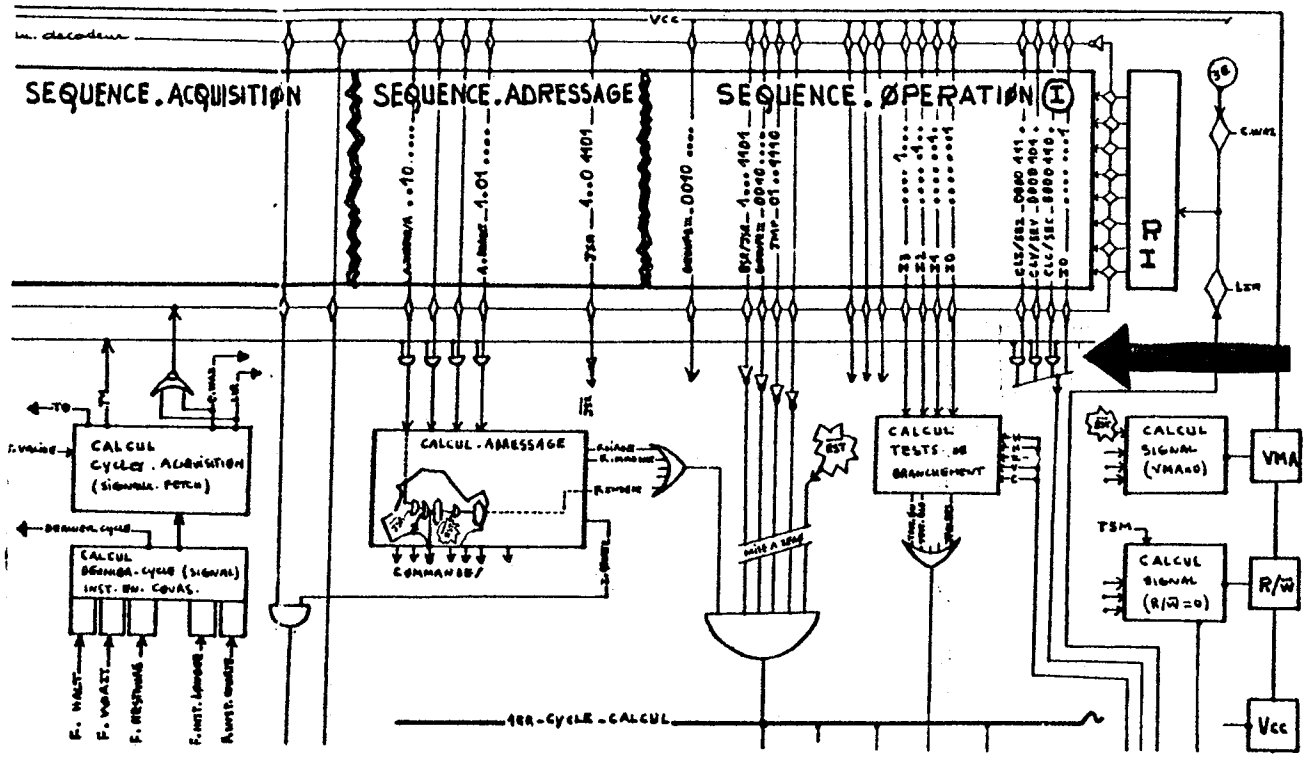


FIG. 62 - Validation des commandes [CD48] par le signal [T2].

La correction de l'anomalie.2 peut être réalisée par la simple validation par le séquenceur (signal T2), des commandes [CD48] permettant la réalisation de la séquence opération de l'instruction CLI.

Ainsi, l'exécution des instructions (CLI, CLV, CLC, SEI, SEV, SEC) n'a lieu ni pendant la séquence d'initialisation, ni pendant la montée du signal RESET.

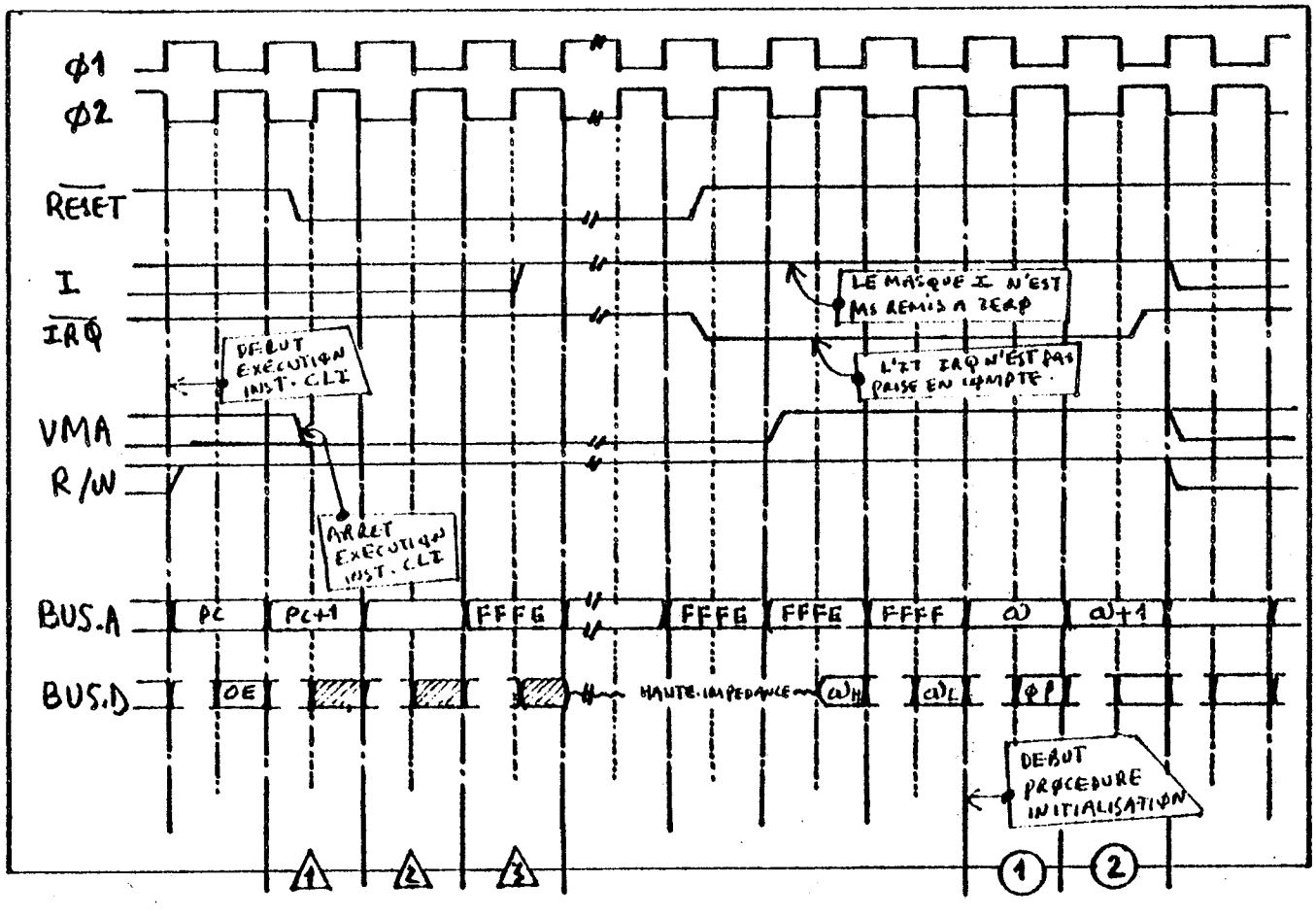


FIG. 63 - Diagramme des temps pour la suppression de l'anomalie.2.

1.3. - Correction de l'anomalie 3

L'anomalie 3 est due à un mauvais traitement de l'instruction CPX (toujours à cause d'une trop grande recherche de facilité dans le câblage).

En effet, l'instruction CPX réalise l'opération de comparaison sur 16 bits, en deux étapes:
- d'abord la comparaison des octets de poids fort (XH-MH),
- ensuite la comparaison des octets de poids faible (XL-ML).

Les bits (N) et (V) du registre des codes conditions RCC sont positionnés seulement en fonction de la comparaison des octets de poids fort. Comme la retenue de l'opération de comparaison des octets de poids faible n'est pas utilisée pour la comparaison des octets de poids fort, le positionnement des bits (N) et (V) se trouve erroné.

La suppression de cette anomalie consiste à forçer les bits (N) et (V) à une valeur logique pendant l'exécution de l'instruction CPX, en le signalant dans les caractéristiques du jeu d'instructions du 6800S.

Ceci peut constituer une différence au niveau du logiciel entre le 6800S et la version actuelle du 6800.

Une solution plus simple et moins coûteuse, consiste à ne pas modifier le câblage, mais simplement signaler cette anomalie d'une manière claire, au niveau du jeu d'instructions.

Une fois connue de l'utilisateur, l'anomalie 3 ne présente pas de conséquences graves dans les applications.

2. - Mécanisme de détection de pannes par codes invalides

2.1. - Caractéristiques générales

Le 6800S peut être considéré comme une nette amélioration de la version actuelle des microprocesseurs 6800. En effet, tous les défauts de la version actuelle du 6800 sont corrigés. De plus, le 6800S serait doté d'un mécanisme de détection de pannes très efficace [13] (efficacité due au grand nombre de codes invalides dans le jeu d'instructions du 6800).

Ceci peut être réalisé par l'adjonction au niveau du circuit intégré d'un bloc logique permettant la détection de l'ensemble des codes invalides.

Ce mécanisme de détection de pannes par codes invalides constitue un nouveau bloc spécialisé du séquenceur : Bloc [ANOMALIE].
Ainsi, toute tentative d'exécution d'un code invalide, soit par erreur de programmation, soit par panne matérielle (collage externe ou au niveau du registre instruction), est immédiatement détectée par ce mécanisme interne.

Une fois l'exécution d'un code invalide détecté, le bloc [ANOMALIE] prend le contrôle du microprocesseur comme suit :

a - avertir l'environnement extérieur par un signal qui serait réservé à cet effet,

b - attendre la fin de l'exécution du code invalide (voir la suppression du bouclage d'exécution dû aux codes invalides 3C, 3D, 9D, DD),

c - mettre le microprocesseur en mode "ATTENTE D'INTERRUPTION" par forçage dans le registre instruction de l'instruction WAI (3E).

Toute la souplesse est laissée au système qui peut ainsi décider de la suite des opérations à effectuer. Cette solution laisse à l'utilisateur l'usage de certains codes invalides intéressants (voir Annexe.2).

Les broches (35) et (38) non utilisées sur le 6800, sont utilisées sur le 6800S pour informer, plus encore, l'environnement extérieur sur l'état interne du microprocesseur:

- détection des codes invalides,
- début de l'exécution d'une instruction.

Le fonctionnement du séquenceur se trouve partiellement modifié par rapport à celui du 6800. L'organigramme suivant présente les caractéristiques générales.

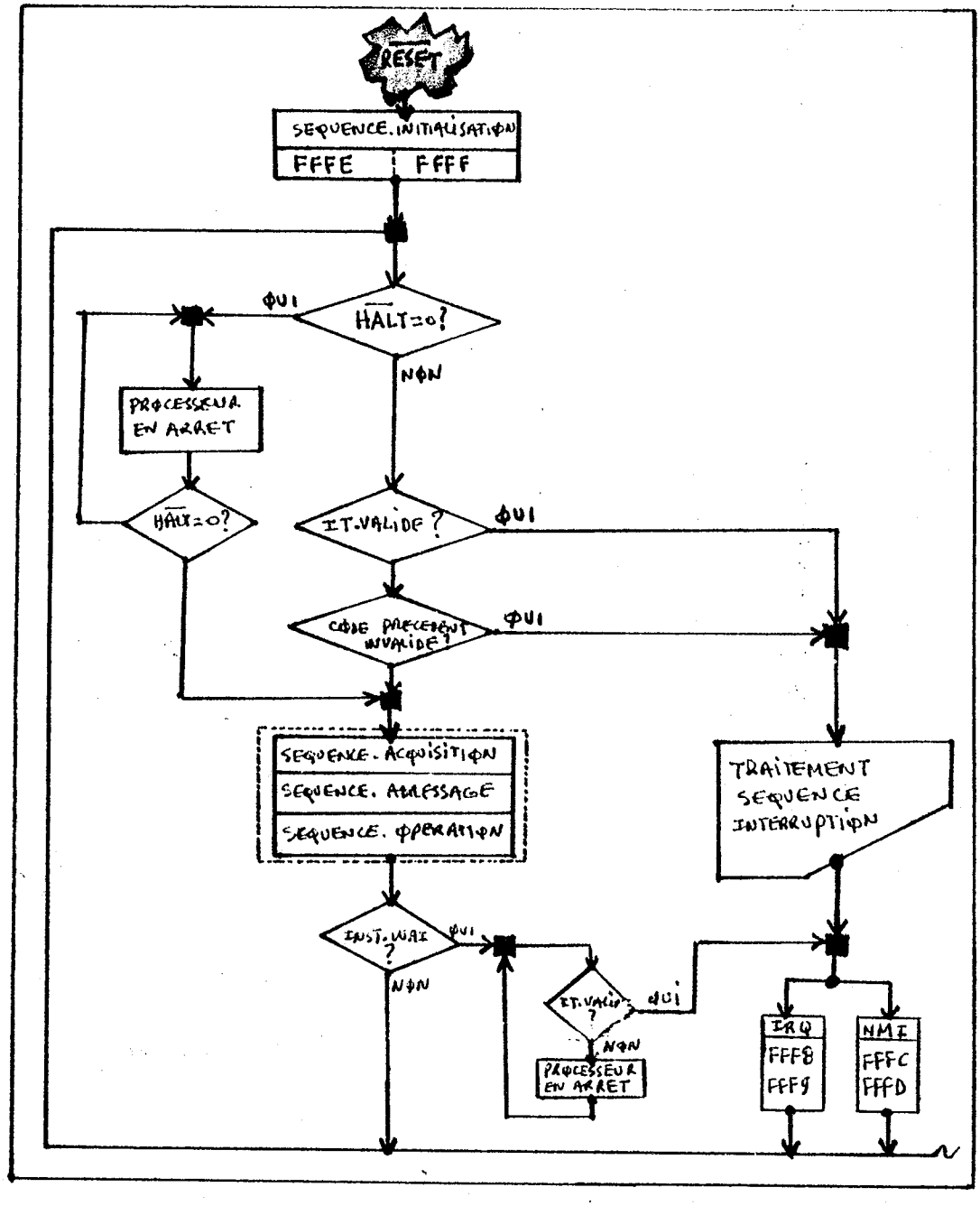


FIG.64 - ORGANIGRAMME GENERAL DU FONCTIONNEMENT DU 6800S .

2.2. - Suppression du bouclage d'exécution par codes invalides

Le schéma du fonctionnement global du séquenceur (FIG.64) suppose que l'exécution de chaque code invalide a une durée infinie. Or, on sait que l'exécution des codes invalides (3C, 3D, 9D, DD) a une durée infinie (bouclage). Donc, il est indispensable de supprimer le bouclage dans l'exécution de ces quatre codes invalides pour permettre au bloc [ANOMALIE] du séquenceur de prendre le contrôle du 6800S.

La suppression du bouclage pour les deux codes invalides (9D, DD) se réalise très facilement par la modification de l'expression booléenne du monome (DC120) du décodeur :

- pour la version actuelle du 6800,
  $DC120 = (I7.I3.I2.\overline{I1}.I0)$

- pour le 6800S,
  $DC120 = (I7.\overline{I5}.I3.I2.\overline{I1}.I0)$

La nouvelle exécution de ces deux codes invalides se résume à l'opération de test [M.(FF)(*)] tout en supprimant la désactivation accidentelle du bloc [OPERATION.I] et par là même, le bouclage de l'exécution.

(*) Valeur hexadécimale.

2.3. - Achitecture du bloc (ANOMALIE)

Le bloc [ANOMALIE] du séquenceur est bâti autour d'un PLA qui permet la reconnaissance de l'un quelconque des 59 codes invalides.

Ce bloc contrôle également la génération des signaux suivants :

- [$\overline{M1}$] , signal qui indique le premier cycle de l'instruction en cours [$\overline{M1}$=0],

- [$\overline{D}$] , signal qui indique la détection d'un code invalide [$\overline{D}$=0].

Les différentes évaluations (électrique, topologique), ainsi que les caractéristiques détaillées de ce bloc sont présentées dans l'Annexe.3.

La description DELTA du bloc [ANOMALIE] est quant à elle présentée dans la figure (66).

La suppression du bouclage pour les deux codes invalides (3C, 3D) s'effectue au niveau du bloc [OPERATION.II].

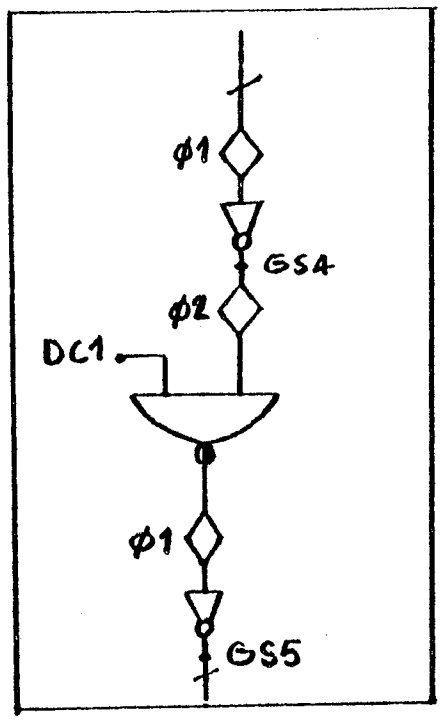


FIG.65A:

- sur la version actuelle du 6800 .

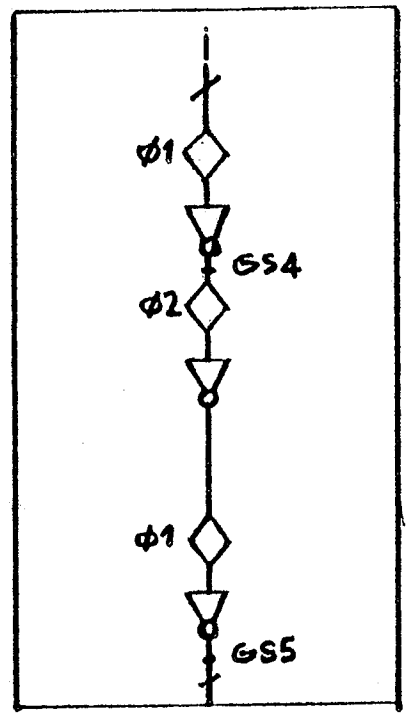


FIG.65B:

- pour le 6800S

La suppression de la condition [DC1 = I1] dans le registre à décalage [SAUVEGARDE] (version actuelle du 6800), permet d'assimiler le séquencement des codes invalides (3C, 3D) à celui des instructions WAI et SWI. Ceci permet également la suppression du bouclage dû à la désactivation accidentelle du registre à décalage [SAUVEGARDE] au niveau de l'étage GS5 (version actuelle du 6800).

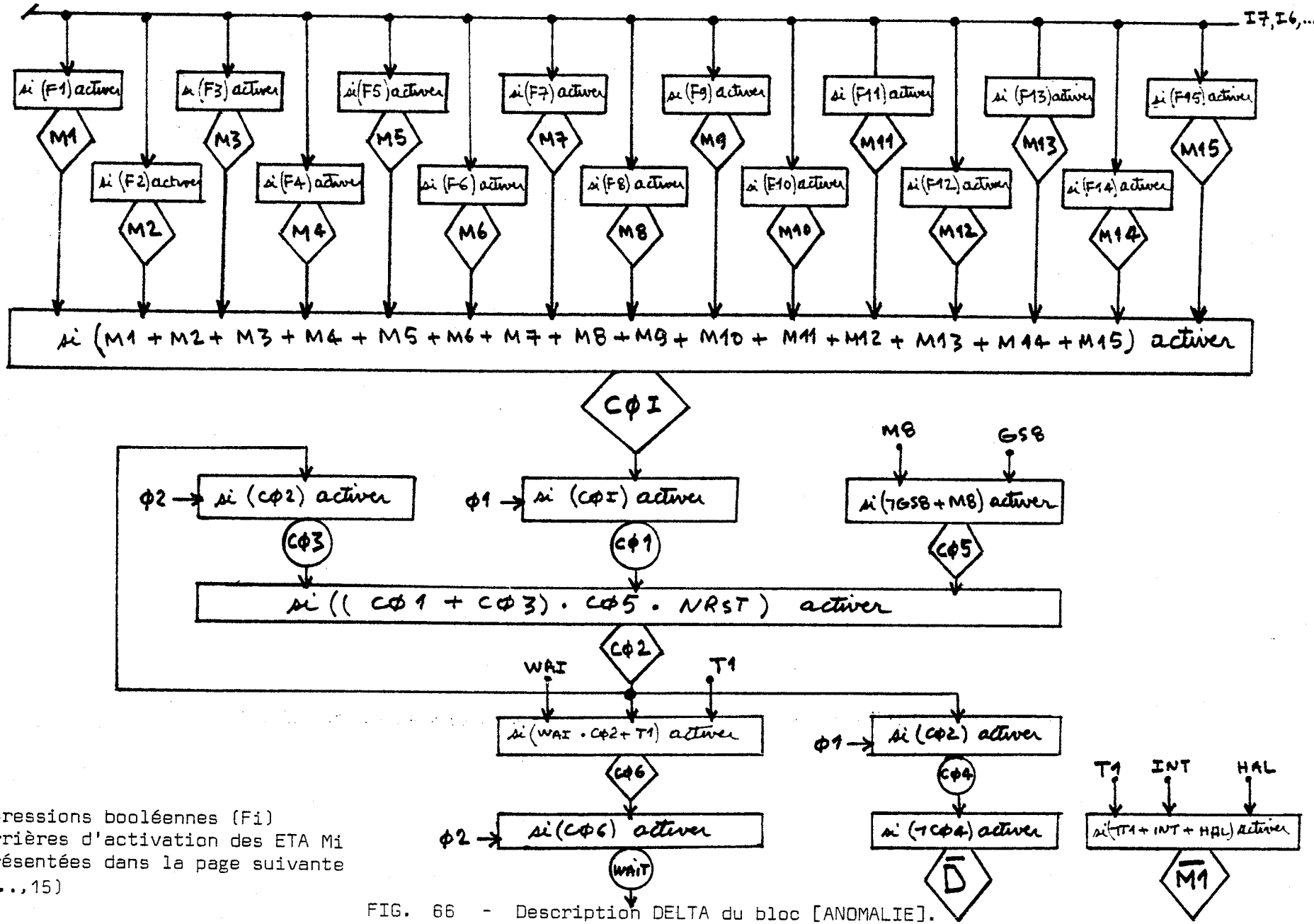


FIG. 66 - Description DELTA du bloc [ANOMALIE].

Note: Les expressions booléennes (Fi) des barrières d'activation des ETA Mi sont présentées dans la page suivante (i=1,...,15)

Nous présentons dans ce qui suit, la description DELTA du bloc [ANOMALIE] chargeable en machine.

```
M1   :=  ¬(I7+I6+I5 I4+I3+I2+I1+I0) ;
M2   :=  ¬(I7+I6+I5+I3+I2+I1) ;
M3   :=  (¬(I7+I6+I5+I3).I2.(¬I1)) ;
M4   :=  (¬(I7+I6+I5).I4.I3.I2) ;
M5   :=  (¬(I7+I6).I4.I3.(¬I2).(¬I0)) ;
M6   := (¬(I7+I6).I5.(¬(I4+I3+I2+I1)).I0) ;
M7   := (¬I7.I6.(¬I3).(¬I2).I0) ;
M8   := (¬(I7+I6).I5.I4.I3.I2.(¬I1)) ;
M9   := (¬I7.I6.(¬(I3+I2).I1.(¬I0)) ;
M10  := (¬I7.I6.I3.I2.(¬I1).(¬I0)) ;
M11  := (¬I7.I6.(¬I5).I3.I2.I1.(¬I0)) ;
M12  := (I7.(¬(I3+I2)).I1.I0) ;
M13  := (I7.(¬(I5+I4)).I2.I1.I0) ;
M14  := (I7.(¬(I6+I5)).I4.I3.I2.(¬I1).I0) ;
M15  := (I7.I6.I3.I2.(¬I1)) ;


COI   := (M1+M2+M3+M4+M5+M6+M7+M8+M9+M10+M11+M12+M13+M14+M15) ;
CO5   := (¬GSB +M8) ;
<φ1>  CO1   <=  COI ;
<φ2>  CO3   <=  CO2 ;
CO2   := ((CO1+CO3).CO5.NRST) ;
CO6   := (T1.CO2+WAI) ;
<φ2> FWAI  <=  CO6 ;
<φ2> CO4   <=  CO2 ;
CO    :=  ¬CO4 ;
CM1   :=  (¬T +INT+HAL) ;
```

Notation :

+	opérateur logique OU
.	opérateur logique ET
¬	opérateur logique NON
Ii	bit (i) du code opération d'une instruction

CONCLUSION

Le présent mémoire débute par la définition d'un formalisme de description qui allie à travers ses primitives de base, l'aspect architectural et comportemental des objets. Cela nous évite d'avoir à vérifier, à chaque fois, l'équivalence entre la description de la machine que l'on manipule pendant la conception (comportement) et celle qui sert à la réalisation (Architecture).

Les deux aspects étant liés dans les descriptions DELTA.

Le formalisme DELTA étant un outil de description au niveau des portes logiques, nous avons volontairement éviter d'introduire des paramètres technologiques (ou électriques) dans la définition de primitives de base.

Ainsi, à partir d'une même description DELTA, plusieurs réalisations technologiques sont possibles qui sont fonction de :

. choix d'une technologie pour la réalisation,
. choix des performances électriques (Horlogerie, ...) .

De plus, les descriptions se trouvent alléger de certains détails qui peuvent s'avérer inutiles et génants.

En effet pendant une simulation logique, par exemple, il est inutile de savoir que telle ou telle porte logique possède un retard électrique $\tau 1$ ou $\tau 2$, ceci ne fait qu'alourdir le traitement. De même, les concepts de base du formalisme sont indépendants de tel ou tel modèle particulier d'architecture et permettent la description des fonctions de contrôle (partie contrôle) aussi bien que celle des opérateurs (partie opérative) dans un circuit séquentiel. Ceci préserve l'efficacité du formalisme (dans le sens de la souplesse d'utilisation) et par la même, supprime l'un des inconvénients, de la majorité des langages de description correspondants au niveau logique.

Il est bien gênant de se voir obliger d'utiliser plusieurs outils de description (dans le même niveau) pour représenter les blocs d'un même circuit séquentiel (partie opérative, partie contrôle) même si leur conception peut se faire séparément.

Malgré les avantages réels que le formalisme DELTA apporte , nous ne prétendons pas avoir défini un formalisme "miracle" qui, à lui seul, supprime tous les problèmes liés à la synthèse automatique dans la conception des circuits séquentiels. Un outil de CAO opérant sur des descriptions DELTA, qui serait mal conçu, aboutirait à des résultats catastrophiques. (coût élevé, risques d'erreurs)...

Dans un contexte VLSI (tendance à une automatisation généralisée), le formalisme DELTA joue pleinement son rôle car il facilite la conception des outils de CAO en permettant la création d'une bibliothèque d'outils de CAO. La compatibilité entre les différents outils de CAO, au sein de cette bibliothèque serait totale car ceux-ci opèrent sur les mêmes structures de données : les descriptions DELTA.

Dans l'approche de la conception à partir des structures "Block.Slice" (30) ou "Mosaïc" (11) (23), un circuit intégré VLSI est obtenu par l'assemblage de blocs spécialisés.

Aussi, le formalisme DELTA permet de réaliser les descriptions de ces blocs spécialisés sur lesquelles s'opérent les différents traitements automatiques.

Nous pensons que tout langage du type RTL peut utiliser facilement et avantageusement le formalisme DELTA pour réaliser les descriptions dans le niveau logique.

Notons cependant que les descriptions DELTA ne constituent que des données pour la synthèse automatique. Tous les mécanismes et toutes les stratégies du traitement automatique doivent être insérer dans les outils de CAO, d'où l'intérêt d'avoir des outils de CAØ compatibles : Les résultats du traitement

d'un outil de CAO (Ti) peuvent être utilisés, de manière directe, comme des données pour un autre outil de CAØ (Tj) sans nécessiter aucun interfaçage supplémentaire.

La dernière partie de ce mémoire, présente une étude approfondie du microprocesseur 6800, ce qui nous permet d'illustrer quelques problèmes généraux liés aux techniques de la conception manuelle. Après avoir présenter les différentes caractéristiques de ce circuit, nous nous intéressons plus particulièrement à ses anomalies de fonctionnement.

En effet, des anomalies de fonctionnement de ce circuit, inconnues de son fabricant et de ses utilisateurs, sont mises à jour. Des solutions sont proposées qui, d'une part corrigent tous les défauts du 6800, d'autre part le dote d'un mécanisme de détection de pannes matérielles très efficace basée sur les codes invalides.

Ceci constitue le projet 6800S dont les détails des différentes évaluations (logiques, électriques, topologiques) sont présentées en annexe.3.

La concrétisation de ce projet permet la réalisation d'une version nettement améliorée du microprocesseur 6800 : le 6800S.

Notons qu'une description DELTA de la partie contrôle complète des microprocesseurs 6800/6800S a été réalisée pour tester les capacités de descriptions du formalisme DELTA (Annexe-1).

ANNEXES

ANNEXE-I : DESCRIPTION DELTA DE LA PARTIE CONTROLE 6800/6800S

Nous présentons dans cette annexe la description DELTA complète de la partie contrôle des microprocesseurs6800 et 6800S. Pour une question de clarté, la structure en blocs du séquenceur a été maintenue. De même, la plupart des notations utilisées dans les synoptiques de la partie contrôle et celui de la partie opérative ont été maintenues.

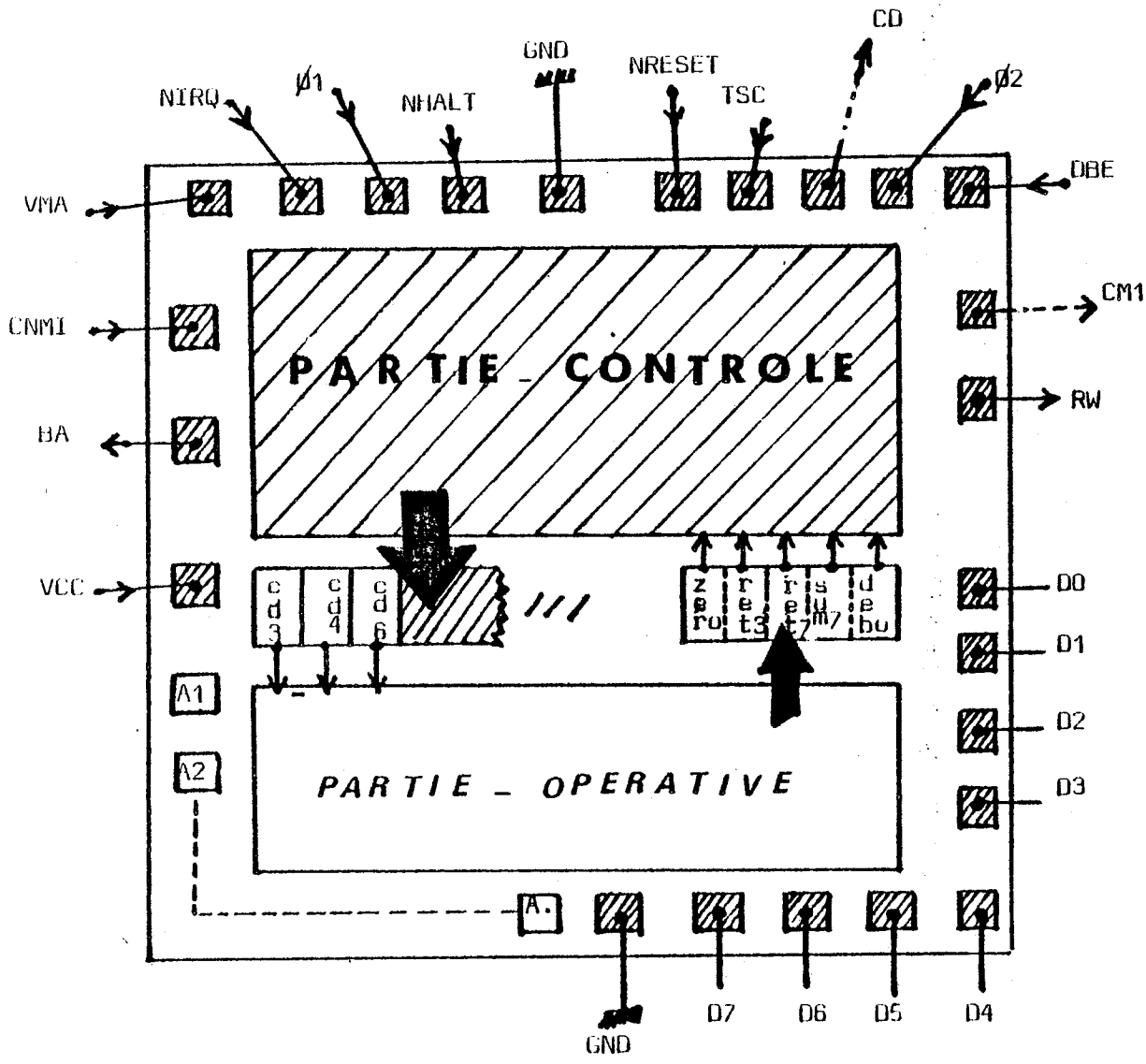


Fig. 67 Représentation des commandes externes à la partie contrôle

Les différents signaux qui arrivent sur la partie contrôle constituent des paramètres pour la description DELTA de celle-ci.

Ce sont :

Commandes venant de la partie opérative :

DEBØ Sortie de l'UAC pour indiquer le débordement,
SUM7 Sortie de l'UAC pour indquer le signe
RET3 Sortie de l'UAC pour indiquer la demi-retenue,
RET7 Sortie de l'UAC pour indiquer la retenue,
ZERO Sortie de l'UAC pour indiquer le résultat nul
R7IH Retenue sortante de INCL vers INCH
DBO,...,DB7 Bus de données de la partie opérative.

Commandes, venant de l'environnement extérieur :

NRESET Commandes d'initialisation,
NHALT Commande d'arrêt,
CNMC Interruption non masquable,
NIRQ Interruption masquable,
DBE Validation du transfert du bus de donnée externe (D),
TSC Commande de mise à haute impédance des bus externes
Ø1, Ø2 Les deux phases de l'horloge,
VCC, GND, la source d'alimentation(tension, masse)
DO,...,D7 bus de données externes

Tous ces paramètres doivent être initialiser par l'utilisateur. Nous présentons dans ce qui suit la description DELTA de la partie contrôle du 6800/6800S.

La description DELTA de la partie opérative peut être réalisée facilement à partir du synoptique de la figure (31).

Note : Les paramètres NRESET, NHALT, CNMI et NIRQ sont actifs au niveau logique zéro.

1.- DESCRIPTION DELTA DU SEQUENCEUR :

* REGISTRE INSTRUCTION (R.I)*

* Description DELTA des bits (0,6,7) du registre instruction (RI)*
* où (j) représente l'indice du bit considéré *

<Ø2> RIj0 ← Dj ;
<LEC> RIj1 ←(RIj0. NWAIT)
RIj2 := (LEC.RIj1.(¬ NLEC) + NLEC. RIj5. (¬LEC) + LEC.NLEC.RIj1.RIj5) ;
RIj3 :=¬TRIj2 ;
NIj := (NLEC.RIj3) ;

RIj4 := ¬RIj3 ;
Ij := (NLEC. RIj4) ;
<NLEC> RIj5 ← RIj4 ;

*Description DELTA des bits (1,2,3,4,5) du registre *
*Instruction ou (R) représente l'indice du bit considéré *

<Ø2> RIk0 ← Dk ;
<LEC> RIk1 ← (RIk0 + WAIT)
RIK2 := (LEC.RIk1. (¬NLEC) + NLEC.RIk5. (¬ LEC) + LEC. NLEC.RIk1.RIk5) ;
RIk3 := ¬RIk2 ;
NIk := (NLEC.RIk3) ;

RIk4 := ¬ RIk3 ;
Ik := (NLEC.RIk 4) ;
<NLEC> RIk5 ← RIk4 ;

* Décodeur instruction *

```
DC 40    :=  ¬ I7 ;
DC 116   :=  ¬ I7 ;
DC 61    :=    (I7.I6) ;
DC 47    :=    I7.(¬I6) ;
DC 63    :=    (I7. (¬I6). (¬I5)) ;
DC 87    :=    (I7. (¬I6). (¬I5)) ;
DC 91    :=    (I7. (¬I6). (¬I5)) ;
DC 122   :=    (¬I7.(¬I6). T2) ;


DC 118   :=    I4 ;
DC 113   :=    (I7.I4. T2)
DC 72    :=    (I5. I4) ;
DC 106   :=    (I5. I4. T2) ;
DC 99    :=    (I7. I5. I4) ;
DC 112   :=    ((¬I7). (¬I6). I5.I4. T2) ;
DC 107   :=    (I7.(¬I5). I4. T2) ;
DC 52    :=    ((¬I7). (¬I5). I4) ;


DC57     :=    ((¬I7). I6. (¬I5). I4) ;
DC46     :=    (¬ (I7 + I6 + I5) - I4);
DC86     :=    (I7. (¬ I4)) ;
DC110    :=    (I5. (¬I4)) ;
DC78     :=    (I7. I5. (¬I4)) ;
DC53     :=    ((¬I7). (¬I6). IS.(¬I4)) ;
DC101    :=    ((¬I7). (¬I6). I5. (¬I4)) ;
DC121    :=   ((¬I7) . (¬I6). I5.(¬I4)) ;
DC98     :=    (I7. (¬I5). (¬I4)) ;
DC123    :=    (I7. (¬I5). (¬I4). T2) ;
DC3      :=    I3 ;
```

```
DC 95    :=  ¬I3 ;
DC 66    :=   ((¬I7). I6. (¬I3)) ;
DC 2     :=   I2 ;


DC 67    :=   ((¬I7). I6. I2) ;
DC 73    :=   (I7. (¬I6). I5. I4. I2) ;
DC 69    :=   (I3. I2) ;
DC 45    :=   (I7. I3.I2) ;
DC 90    :=   (I7. I3.I2) ;
DC 24    :=   (¬I3. I2) ;
DC 26    :=   (¬I7. I6. (¬I3). I2) ;
DC 17    :=   (¬I7. I6. (¬I3). I2) ;


DC 80    :=   (I3. (¬I2)) ;
DC 8     :=   (¬I3. (¬I2)) ;
DC 1     :=   I 1 ;
DC 77    :=   (I5. (¬ I4. I1. T2) ;
DC 42    :=   (I3. I2) ;
DC 44    :=   (I3. I2) ;
DC 25    :=   (I3. I2. I1) ;
DC 68    :=   (I3. I2. I1) ;
DC 124   :=   (I7. (¬I6). I3. I2. I1) ;
DC 6     :=   (¬I7). (¬I6). (¬I5). (¬I4) . I3. I2. I1);
DC 125   :=   (¬I3. I2. I1) ;
DC 65    :=   (¬I7 - (¬I6). I5. I4. (¬I3). I2. I1. T2) ;
DC 14    :=   (¬I7. (¬I6). (¬I5). I4. I3. (¬I2). I1) ;


DC 7     :=   (¬(I7 + I6 + I5 + I4). I3. (¬I2). I1) ;
DC 127   :=   (¬I3. (¬I2). I1) ;
DC 128   :=   (¬I3. (¬I1)) ;
DC 5     :=   (¬(I7 + I6 + I5 + I4). I3.I2 (¬I1)) ;
DC 82    :=   (I3. ¬(I2 + I1)) ;
DC 10    :=   (¬I7. I6. I3. (¬(I2 + I1))) ;
DC 16    :=   (¬I7. I6. I3. (¬(I2 + I1))) ;
```

```
DC 9     :=   (¬(I7 + I6 + I5). I4. I3. (¬I2 + I1))) ;
DC 102   :=   (¬(I7 + I5). T2) ;
DC 79    :=   (¬(I7 + I6 + I5 + I4). I3. (¬(I2 + I1)));
DC 92    :=   (¬(I7 + I6 + I5 + I4). I3. (¬(I2 + I1))) ;
DC 103   :=   (¬(I7 + I6 + I5 + I4). I3. (¬(I2 + I1))) ;
DC 129   :=   (¬(I7 + I6 + I5 + I4). I3. (¬(I2 + I1)). T2) ;
DC 28    :=  ¬(I3 + I2 + I1) ;

DC 4     :=   I0 ;
DC 114   :=   I0 ;
DC 11    :=   (I3. I0) ;
DC 27    :=   (¬I7. I6. I3. I2. I0) ;
DC 127   :=   (I3. (TI2). I0) ;
DC 13    :=   (I7. I3. (¬I2). I0) ;
DC 64    :=   (I7.I2.I1.I0) ;
DC 88    :=   (I7.I2.I1.I0) ;

DC 62    :=   (I7. I3.I2.I1.I0) ;
DC 71    :=   (I7. I6.I3.I2.I1.I0) ;
DC 97    :=   (¬I3. I2. I1. I0) ;
DC 93    :=   (I7. (¬I3) .I2. I1. I0) ;
DC 51    :=   (I6. I6. (¬I3). I2. I1. I0) ;
DC 37    :=   (I7. (¬I6). (¬I3). I2. I1. I0) ;
DC 50    :=   (¬(I7 + I6). I5.I4. (¬I3) + I2. I1. I0. T2) ;
DC 21    :=   (¬(I7 + I6 + I5 + I4.I3). I2. I1. I0. T2) ;

DC 20    :=   (¬(I7.I6). I5. I4.I3. (TI2). I1.I0) ;
DC 58    :=   (¬I3 + I2). I1. I0) ;
DC 48    :=   (¬(I3 + I1). I0) ;
DC 55    :=   (¬(I3 + I1). I0) ;
DC 104   :=   (I7. I3.I2. (¬I1). I0) ;
DC 111   :=   (I7. I3. I2. (¬I1). I0) ;
DC 120   :=   (I7. I3. I2. (¬I1). I0) ;
```

```
DC 130   :=   ( I7. I6. I5. I3. I2. (¬I1). I0) ;
DC 117   :=   (¬I4. I3. I2. (¬I1). I0) ;
DC 54    :=   (I7. ¬(I5 + I4). I3. I2. (¬I1). I0) ;
DC 109   :=   (I7. ¬(I5 + I4). I3. I2. (¬I1). I0. T2)
DC 76    :=   (¬I7 + I6). I5.I4. (¬I3). I2.(¬I1). I1. T2) ;
DC 31    :=   (I3. ¬(I2 + I1). I0) ;
DC 74    :=   (¬(I7 + I6 + I5 +I4).I3.( (I2 + I1)). I0. T2) ;
DC 75    :=   (¬(I7 + I5 + I5 + I4). I3 ( (I2 + I1)). I0. T2) ;


DC 94    :=   (I7. I3. I2. (¬I0)) ;
DC 35    :=   (I7. I3. (¬(I2 + I0))) ;
DC 81    :=   (I7. I6. I3. I2. I1. (¬I0)) ;
DC 89    :=   (¬I7. I6. I3. I2. I1. (¬I0)) ;
DC 105   :=   (¬I7. I6. I3. I2. I1 (¬I0)) ;
DC 119   :=   (¬I7. I6. I3. I2. I1 (¬I0)) ;
DC 39    :=   (¬(I7 + I6 + I5 + I3). I2. I1. (¬I0)) ;
DC 41    :=   (¬(I7 + I6). I5. I4. (¬I3). I2.I1 (¬I0). T2) ;


DC 18    :=   (¬(I7 + I6 + I5 + I4 + I3). I2. I1. (¬I0). T2) ;
DC 48    :=   (¬(I3 + I2). I1. (¬I0)) ;
DC 30    :=   (I7. (¬ (I3 + I2)). I1 (¬I0)) ;
DC 29    :=   (I3. I2. ( ¬(I1 + I0))) ;
DC 34    :=   (I7. I3. I2. (¬(I1 + I0))) ;

DC 70    :=   (I7. I3. I2 (¬(I1 + I0))) ;
DC 84    :=  ¬ (I3 + I2 + I1 + I0) ;
DC 152   :=  ¬ (I7 + I6 + I5 + I4) ;
DC 132   :=   (I7. I3. I2) ;
DC 133   :=   (I7. I3. I2) ;
DC 153   :=  ¬(I7 + I6 + I5 + I4) ;
DC 482   :=   (¬(I7 + I6 + I5). I4 (¬I3). I2. I1 (¬I0)) ;
DC 493   :=   (¬(I7 + I6 + I5). I4. (¬I3). I2. I1. (¬I0)) ;
DC 382   :=   (¬I7. I6. (¬(I5 + I4))) ;
```

```
* BLOC (ACQUISITION) *

 ADER :=  (GR8 + MEX2. DC105 + MX2. (DC105 + DC104 + DC101) + SF13 + SF28 ;
<Ø2>      SFO3 ←ADER
<Ø1>      T1  ← (NRST.SFO3 +STOP) ;
<Ø2>      SFO7← T1 ;
<Ø2>      SFO8← NHAL ;
<Ø1>      SF10← (SFO8-SFO7) ;
 T2   :=  (NRST. SF10. NHAL) ;
 NT2  := ┐(LEC + T2) ;


 NTO1 := ┐T1 ;
 CØNE :=  (HAL + NTO1 + Ø1) ;
 LEC  := ┐CONE ;
 NLEC := ┐LEC ;


 WAI  := ┐ (NT1 + NINIT) ;
<Ø2>      WAIT ← WAI ;
 * WAIT est défini autrement pour le 6800 S/voir bloc (ANOMALIE) *
 NWAIT := ┐WAIT ;
 PL12  := (DC95.GS2 + (┐(DC97 + DC96). (┐DC 116)). CAL1) ;
 SF21  := (DC94 + DC93). CAL2 ;

<Ø2>  SF22 ←CAL2 ;
<Ø2>  SF24 ←(DC92 + DC91 + DC90) ;
<Ø1>  SF26 ←(SF22.SF24.NRST) ;
 SF28 :=  (SF21 + PL12 + (┐DC94. SF26)) ;
 SF30 :=  (┐DC98 - (SF98 + SF13) ) ;
 SF34 :=  (NBRA - DC101.MX2) ;
 SF32 :=  ((DC98 + DC99). GS3) ;
 PLN1 :=  (SF30 + SF32 + SF34) ;
```

BLOC (ADRESSAGE)

```
<Ø2>     DX 00 ← GS3 ;
 DX04    := (DC110 + DC109) ;
<Ø2>     MX0 ←DX04 ;
<Ø1>     DX08← (DC117.DX00 + ( DC117 . MX0)) ;
 MX1     := (DX08.NRST) ;
<Ø2>     DX09← MX1 ;
<RST>    DX11← GND ;
 DX10    := (Ø2.DX09.(¬RST) + RST. DX11. (¬Ø2) + Ø2.RST. DX09 - DX11) ;
<Ø1>     MX2 ← DX10 ;


<Ø2>     MEX1← (¬DC17 - DC106) ;
<Ø1>     MEX2← MEX1 ;
<Ø2>     EXT04←  (DX04 + MEX2) ;
 PS04    := (DC 111 - EXT04) ;


 MD10    :=  DC107 ;
```

* BLOC (OPERATION-I) *

```
 CA16      :=  (MX2 + MEX2 + DC107) ;
<Ø2>          SOP1 ← CA16 ;
<Ø1>          SOP2 ← SOP1 ;
CAL1      :=  (¬(¬(SOP2 + DC124 + DC 122) + DC111 + DC120 + DC121 + RST)) ;
<Ø2>          SOP4  ← CAL1 ;
<Ø1>          CAL2  ← SOP4 ;
```

* BLOC (OPERATION-II)*

```
<Ø1>    SRO ← PSO3 ;
 GSO    := ((SRO + DC 112). NRST) ;
<Ø2>    SR3 ← GSO ;
<Ø1>    GS1 ← SR3 ;
<Ø2>    SR8 ← GS1 ;
<RST>   SR1O  GND ;
 SR9    := (Ø2.SR8 (¬RST) + RST. SR1O. (¬Ø2) + Ø2. RST. SR1O. SR8) ;
<Ø1>    GS2  ←(¬DC1O8. SR9) ;


<Ø2>    SR12 ←GS2 ;
<RST>   SR14 ←GNO ;
 SR13   := (Ø2.SR12.(¬RST) + RST.SR14 (¬Ø2) + Ø2. RST. SR12.SR14) ;
<Ø1>    GR3 ←(SR13. DC8O) ;


<Ø2>    SR17 ←GR3 ;
<Ø1>    GR4 ← SR17 ;
<Ø2>    SR18 ← GR4 ;
<   >   SR2O ←GND ;
 SR19   := (Ø2.SR18(¬RST) + RST.SR2O (¬Ø2) + Ø2. RST. SR2O.SR18) ;


<Ø1>    GR5 ← SR19 ;
<Ø2>    SR22← GR5 ;
<Q1>    GR6 ← SR22 ;
```

```
<Ø2 >   SRO7 ←GS8 ;
<Ø2 >   SR23 ←GR6 ;
<Ø2 >   SRO6 ←(GS1 - DC82) ;
<Ø1 >   SR25 ←(SRO7 + SR23 + SRO6) ;
 GR7    :=  (SR27 - NRST) ;
<Ø2 >   SR26 ←GR7 ;
<Ø1 >   GR8  ←(SR27 - DC116 + SR26) ;
<Ø2 >   SR31 ←GS2 ;
<Ø1 >   GS3  ←(SR31. DC16) ;
<Ø2 >   SR33 ←GS3 ;
<RST>   SR35 ←GND ;
 SR34   := (Ø2.SR33- (¬RST) + RST-SR,35.(¬Ø2) + Ø2.RST. SR33.SR35) ;
<Ø1 >   GS4  ←(SR34.DC118) ;
<Ø2 >   SR38 ←(GS4 ;
<Ø1 >   GS5  ←(SR38-DC1) ;
<Ø2 >   SR40 ←GS5 ;
<RST>   SR42 ←GND ;


 SR41   := (Ø2.SR40.(¬RST) + RST.SR42.(¬Ø2) + Ø2.RST. SR40.SR42) ;
<Ø1 >   GS6  ←SR41 ;
<Ø2 >   SR44 ←GS6 ;
<Ø1 >   GS7  ←(SR44 + SR48) ;
<Ø2 >   SR51 ←GS7 ;
<Ø2 >   INT2 ←NINT
<Ø2 >   SR47 ←(¬DC4.GST) ;
 SR48   := (SR47. (NRST.INT2)) ;
<Ø1 >   GS8 ←(SR51.DC4 + (¬INT2. SR47) + RST) ;
```

BLOC (ANOMALIE) / 6800S

```
M1    :=  ¬(I7 + I6 + I5 + I4 + I3 + I2 + I1 + I0) ;
M2    :=  ¬(I7 + I6 + I5 + I3 + I2 + I1) ;
M3    :=  ¬(I7 + I6 + I5 + I3 + I1). I2) ;
M4    :=  ¬(I7 + I6 + I5) . I4. I3. I2) ;
M5    := ( ¬( I7 + I6 + I2 + I0). I4. I3) ;
M6    := ( ¬( I7 + I6 + I4 + I3 + I2 + I1) I5. I0) ;
M7    := ( ¬( I7 + I3 + I2) . I6 . I0) ;
M8    := ( ¬( I7 + I6 + I1) . I5 . I4 . I3. I2) ;
M9    := ( ¬( I7 + I3 + I2 + I0) . I6 . I1) ;
M10   := ( ¬(I7 + I1 + I0). I6. I3. I2) ;
M11   := ( ¬(I7 + I5 + I0). I6. I3. I2. I1) ;
M12   := ( ¬(I3 + I2) . I7. I1. I0) ;
M13   := ( ¬(F5 + I4). I7. I2. I1. I0)' ;
M14   := ( ¬(I6 + I5 +I1). I7. I4. I3. I2. I0) ;
M15   := (¬ I1.I7.I6.I3.I2) ;


CO1   := (M1+M2+M3+M4+M5+M6+M7+M8+M9+M10+M11+M12+M13+M14+M15) ;
<Ø1>  := CO1 ←CO1 ;
<Ø2>  := CO3 ←CO2 ;
CO5   := (¬GS8 + M8) ;
CO2   := ((CO1 + CO3). CO5. NRST) ;


CO6   := (T1.CO2 + WAI) ;
<Ø2>    WAIT ← CO6 ;
<Ø2>    CO4 ← CO2 ;
CD    := ¬ CO4 ;
CM1   :=  (¬T1 + INT + HAL) ;
```

MEMORISATION (REST)

```
<Ø2>      RSTO  ← ┐NRESET ;
<Ø1>      RST1  ←  RST4 ;
 RST2 := (Ø2.RSTO.(┐Ø1) + Ø1.RST1(┐Ø2)+ Ø1.Ø2.RSTO.RST1) ;
<┐RST2> RST4 ← GNO <┐O1.RSTO>    RST4 ← VCC ;


<Ø1>      RST  ← RST4 ;
 NRST := ┐RST ;
 V A2  :=  (NRST.NRESPT) ;
```

MEMORISATION (HALT)

```
<Ø1>  HLTO ← NHALT ;
<Ø2>  HLT1 ← (ADER.HLTO) ;
<Ø1>  HLT4 ← HLT4 ;
 HLT6 := (Ø1.HLT5. (┐Ø2) + Ø2. HLT1 (┐ Ø1) + Ø1.Ø2.HLT1. HLT5) ;


<Ø2>  HLT8 ←┐HLTO ;
<Ø1>  HLT2 ←┐HLT4 ;
 HLT3 := (Ø1.HLT2.(┐Ø2) + Ø2. HLT8 (┐Ø1) + Ø1.Ø2.HLT2. HLT8) ;
<HLT3 + RST> HLT4 ← GND, <HLT 6>   HLT4 ←  VCC ;
<Ø1>  HLT7  ← HLT4 ;
 HAL  := ┐(HLT7 + RST) ;
 NHAL := ┐ HAL ;
```

* MEMORISATION ($\overline{NMI}$)*

<Ø2> NMI1 ←CNMI ;
<Ø1> NMI2 ←¬NMI4 ;
NMI3 := (Ø1.NMI2. (¬Ø2) + Ø2.NMI1.(Ø1) + Ø1. Ø2. NMI1.NMI2) ;
<NMI3>NMI4 ← GND,< (Ø1+CNMI)> NMI4 ← VCC ;
<Ø1> NMI5 ← NMI4 ;
<Ø2> NMI6 ←¬NMI5 ;
<Ø1> NMI7 ← (¬NMI9.NMI4.NMI6) ;

<(NMI8 + RST)> NM2I ← GND, <NMI7> NM2I ← VCC ;
<Ø2> NMI8 ← (DC4.NM2I. GS8) ;
<Ø1> NMI9 ← NMI8 ;

MEMORISATION ($\overline{IRQ}$)

<Ø1> IRQ3 ← ¬I2M ;
<Ø2> IRQ4 ← IRQ3 ;
IRQ1 := (IRQ6.NIRQ) ;
<Ø2> IRQ2 ← ¬(IRQ1 + GS8) ;
<I2M> IRQ5 ← GND, <(IAQ1.IRQ2)> IRQ5 ← VCC ;
<Ø1> IRQ6 ←¬IRQ5 ;

IRQ8 := ¬ (¬DC4.GS8 + T1) ;
<Ø2> IRQ9 ← HAL ;
<Ø1> IRQ10 ← IRQ9 ;
STOP := (NHAL.NRST.IRQ10) ;
IRQ11 := ¬ (IRQ8 + STOP) ;

<Ø1> IRQ7 ← IRQ5 ;
INT := ((NM2I + IRQ7). IRQ11) ;
NINT:= ¬ INT ;

* MEMORISATION (NMI)/6800^{S} *

```
<Ø2>      NMI1 ←GNMI ;
<Ø1>      NMI2 ←┐NMI4 ;
 NMI3 := (Ø1.NMI2(┐Ø2) + Ø2.NMI1. (┐Ø1) + Ø1.Ø2.NMI1. NMI2)


<NMI3>    NMI4← GND, < ┐(Ø1 + CNMI)> NMI4 ← VCC ;
<Ø1>      NMI5 ←NMI4 ;
<Ø2>      NMI6  ←  NMI5 ;
<Ø1>      NMI7  ←( ┐NM10. NMI4. NMI6) ;


<NMI9>    NM2I ←GND, <NMI7>  NM2I ← VCC ;
<Ø2>      NMI8 ←(RST + (┐DC4.  GS8. NM2I)) ;
<Ø2>      NMI10 ←  ┐NMI9 ;
<Ø1>      NMI9  ←  NMI8 ;
```

2.- DESCRIPTION DELTA DE L'INTERFACE DE COMMANDE

* MEMORISATION (TSC)*

```
<Ø1>      TSO ← BTE ;
 TS1   := (STC + HAL + TSO) ;
```

* MEMORISATION (DBE)*

```
 BEO   :=  ⌝DBE ;
<Ø1>   BE1  ←  WRI ;
<BEO>  BE2 ← BE1 ;
 BE3   :=  ⌝BE2 ;
<BEO>  BE4 ←GND ;
<BE3>  BE5 ←DBE ;
 BE6   :=(BEO.BE4.(⌝BE3) + BE3. BE5 (⌝ BEO) + BEO. BE3. BE4. BE5) ;
<BEO>  BE7 ←VCC ;
<BE6>  BE8 ←GND ;
 BRW   := (BEO.BE7.(⌝BE6) + BE6. BE8. (⌝BEO) + BEO.BE6.BE7.BE8) ;
```

*SIGNAL (BA) *

```
<Ø2>   BA1 ←GS7;
<Ø2>   BA2 ←⌝NM2I ;
 B TE  := (BA1.BA2.INT2. (⌝IO)) ;
 B A3  := (BTE + HAL) ;

<Ø1>   BA4 ←BA3 ;
<Ø2>   BA5 ←BA4 ;
 BA  := ( ⌝TSC. BA3. BA5) ;
```

*SIGNAL (R/W) *

```
RW      := ((GS2 + GS3 + GS4 + GS5 + GS6). DC68) ;
DEC1    := (RW9 + DC69.GS1) ;
RW5     := (DEC1 + CAL2. (DC62 + DC63) + CAL1. DC64 + DC65) ;
DEC     := (GSO. DC69) ;
WRT4    := (DEC + RW5) ;


<Ø2>    WRI ← WRT4 ;
<Ø1>    RW14 ← WRI ;
RW16    := ( ¬TSI. RW14) ;
RW17    := ¬(TSI + RW16) ;


<RW17>  RW18 ← VCC ;
<RW16>  RW19 ← GND ;


R/W := (RW17.RW18.(¬RW16) + RW16. RW19. (¬RW 17) + RW16. RW17. RW18. RW19);
```

* SIGNAL (VMA) *

```
<Ø2>      VMO  ← DC 129 ;
<Ø1>      VM1  ← VMO ;
<Ø2>      VM2  ← VM1 ;
<Ø2>      VM6  ← ((DC126 + DC127).GSO) ;
<Ø2>      VM9  ← (DC130.CAL2) ;
<Ø2>      VM15 ← ((GS1 + GSO). DC128) ;
<Ø2>      VM19 ← (¬DC116.(GS2 + GS3)) ;

 TIL4     := (GS7 + MX1 + MxO + DC87. CAL1 + DC86. GS3 + CAL16. DC88) ;
<Ø2>      VM26 ← TIL4 ;
<Ø1>      VM29 ← ((VM2 + VM6) + VMO + VM9 + (VM12 + VM15) + VM19 + VM26) ;
 VMA      := ¬(TSC + VM29 + HAL + VA2) ;
```

* BIT SIGNE DU RCC (N)*

```
 Z144     := ( ¬DC153. CAL2) ;
<Ø2>      Z14  ←  Z144 ;
 N6       := ¬(Z14 + DBPS)
<Ø1>      N ←(DB3. DBPS + SUM7. Z14 + N6. N9) ;
<Ø2>      N9 ← N ;
```

```
* MEMORISATION RETENUE (C) *

 DBB2     := (DC26.CAL1) ;
<Ø2>      C1 ← DBB2 ;
<Ø1>      C2 ← C1 ;

<Ø1>      C5 ← DBO ;
<Ø2>      C21← (C5.C2) ;


<Ø2>      C15 ←(CAL2. DC152. DC33) ;
<Ø2>      C10 ←(¬ (RE7F-DC9). DC10 + DC11). CAL2.(¬ DC132). (¬DC152)) ;
<Ø2>      C12 ←(CAL2. DC27) ;
 C13      := (C12 + (¬RET7)) ;
 RE7M     := ¬ C13 ;


<Ø2>      C17 ←DC4 ;
<Ø2>      C19 ←DC5 ;
 C31      := (¬ C17. C19) ;
 RE7A     := (C21 + C10. C13 + RE7M. C15 + C5.DBPS + C28. C30) ;


<Ø1>      RE7 F ← RE7 A ;
<Ø2>      C30   ← RE7 F ;
 C28      := ¬(C3 + C10 + C15 + C19 + DBPS) ;
```

* MEMORISATION DEBORDEMENT (V)*

```
 OU2        :=    (CAL1.DC35) ;
<Ø2>        V1 ← OU2 ;
<Ø1>        V2 ← V1 ;
<Ø2>        V4 ← ¬(DEBO + V2) ;
 V1 A       :=  ¬ (DC16 + DC17) ;
<Ø2>        V6 ←  (Z 144 . V1A) ;
 V8         := ((¬RE7A.SUM7) + (¬SUM7.RE7A)) ;


<Ø2>        V12 ←(¬V1A.CAL2) ;
<Ø2>        V14 ←DC4 ;
 V15        := ¬V14 ;
<Ø2>        V17 ← DC7 ;
<Ø1>        V  ←(V4. V6 + V8.V12 + V15. V17 + DB1. DBPS + V24.V28) ;


<Ø2>        V28 ←V ;
<V24>       := ¬(V6 + V12 + V17 + DBPS) ;
```

* MEMORISATION ZERO (Z)*

```
<Ø2>    ZO ← (DC79.CAL2) ;
 XHD3   := (ZO.NRST) ;
<Ø1>    Z1 ← XHD3 ;
<Ø2>    Z2 ← Z1 ;
<Ø1>    Z3 ← Z2 ;
<Ø2>    Z4 ← Z3 ;
 Z18    :=  (Z14 + DBPS + Z1O + Z4) ;

<Ø2>    Z8 ← (DC9.CAL2) ;
<Ø1>    Z9 ← Z8 ;
<Ø2>    Z1O← Z9 ;
 Z12    := (Z1O.Z21 + Z14) ;


<Ø1>    Z  ← (Z4-RE7M + DB2.DBPS + Z12. ZERO + Z18. Z21).

<Ø2>    Z21 ←Z ;
```

* MEMORISATION MASQUE IRQ (I) *

```
<Ø2>     ITO ←GS8 ;
<Ø1>     IT1 ←DB4 ;
<Ø2>     DBPS← (GS2 - DC2O + DC19) ;
<Ø2>     IT2 ← DC4 ;
 IT3     := ¬IT2 ;
<Ø2>     RDA ← DC6 ;


 IT4     := (ITO + IT1. DBPS + IT3. RDA + IT6. IT5) ;
 I2M     := (DBPS.IT4) ;

<Ø1>     IM ← IT4 ;
<Ø2>     IT5← IM ;
 IT6     := ¬(ITO + DBPS + RDA) ;
```

MEMORISATION DEMI RETENUE (H)

```
<Ø1>     HO ← DB5 ;
<Ø2>     H1 ← RET3 ;
<Ø2>     H2 ← ((DC13 + DC14) . CAL2) ;
 H3      :=   (H2 + DBPS) ;



<Ø1>     H4 ← RE3F ;
```

* CALCUL BRANCHEMENT (NBRANCH) *

```
B2      :=   (¬Z. DC1 + (¬(RE7F + DC1))) ;
BPE8    :=  ¬(DC1.DC2) ;
B4      :=   (Z. DC1 + RE7F. BPE8) ;
BR1     :=   ((¬DC4. DC2.B2 + DC4.B4). (¬DC3)) ;

B19     :=   (¬(DC4 + N) + N.DC4)) ;
B16     :=  (¬(V + DC4) + V. DC4) ;
BRO7    :=   (¬DC1. B16) ;
B2O     :=   (DC1. B19 + BO7) ;
BR3     :=   (DCZ. DC3. B2O) ;
B13     :=   (( Z + BR1O). DC1 + (¬(BRO9 + DC1))) ;
BRO9    :=   (¬(N + V). (¬ (N.V))) ;
BR1O    :=  ¬BRO9 ;
B1O     :=   (¬(Z + BR1O). DC1 + ( ¬ (BR1O + DC1))) ;
BR2     :=   ((¬DC4. B1O + DC4.B13). DC2. DC3) ;

BRO6    :=   (¬DC1 + RE7F + Z) ;
BR4     :=   ( ¬ (DC2 + DC3 + DC4). (¬(BRO6. DC1))) ;

NBRA    :=   (¬(BR1 + BR2 + BR3 + BR4)) ;
```

*COMMANDES (DNC) *

```
 Nφ1      := ¬ φ1 ;
 Nφ2      := ¬ φ2 ;
 C31      :=   (DC1 13 + MX1 + GS8 + MEX2 + GR8) ;
<φ2>      C32 ←C31 ;
<Nφ2>     CD3 := C32/ CD3 := GND ;

*CD3 représente la commande : DB → INCL*

<φ2>      C35 ← C31 ;
<Nφ2>     CD4 := C35/CD4 := GND ;

*CD4 représente la commande : ADL → INCL*
<φ1>      DB71 ← DB7 ;
 TIHB  := ¬(DC54 + DC 59) ;
<φ2>      C15 ←((DB71 + TIHB). RET7) ;
<φ2>      C14 ← DX1 ;
<φ2>      C110 ←(¬RET7. (¬ TIHB). MX1. DB71) ;
<φ1>      C112 ←((C14. C15) + C110) ;


 C125     :=   (C112 + R7IH - C123) ;
<Nφ1>     TN2 :=   C125/ TN2 := GND ;
<NØ1>     TN3 :=   C125/TN3  := GND ;


<φ2>      C115  ←  (1 DC 89 DC87. CA16) ;
<φ1>      C116  ←  C115 ;
<φ2>      C117  ←  TIL4 ;
<φ1>      C118  ←  C117 ;
 C123  := ((C116 + C118 + HAL) + (¬ NTO1. INT)) ;

<Nφ1>     TN13 := C123/ TN13 := GND ;
<Nφ1>     TN14 := C123/ TN14 := GND ;
```

* COMMANDES (PC, TINC) *

SLBC := (¬DC17. DC77 + DC78 - 653 + DC75 + DC76) ;
C76 := ¬(DC113 + GR8 + MEX2 + SLBC + GS8 + GSO) ;
PLN2 := ¬ PLN1 ;
<Ø2> C79 ← (PLN2. C76) ;
<NØ2> CD7 := C79 / CD7 := GND ;

CD7 représente la commande : TINC → AD

<Ø2> C61 ←(MEX2 + T1 + MDIO + MXO) ;
<NØ2> CD6 : = C61/ CD6 : = GND ;

*CD6 représente la commande : TINC → PC *

<Ø2> C81 ← PLN1 ;
<NØ2> CD8 := C81/ CD8 := GND ;

CD8 représente la commande : PCL → ADL

<Ø2> C80 ← PLN1 ;
<NØ2> CDH8 := C80/ CDH8 := GND ;
CDH8 représente la commande : PCH → ADH

<Ø2> C90 ← DEC 2 ;
<NØ2> CD9 := C90/CD9 := GND ;

CD9 représente la commande : PCL → DB

<Ø2> PHD3 ← DEC2 ;
<Ø1> C100 ← PHD3 ;
<Ø2> C101 ← C100 ;
<NØ2> CD10 := C101/CD10 := GND ;
CD10 représente la commande : PCH →DB

COMMANDES (DB)

<Ø2> C541 ←GS8 ;
<NØ2> CD54 := C541/CD54 := GND ;
CD54 représente la commande : 0 →DBO.

<Ø2> C552 ←(¬(RST + DC4). GS8) ;
<NØ2> CD55 := C552/CD55 := GND ;
*CD55 représente la commande : 0 →DB1 *

<Ø2> C563 ← ((DC4 + NM2I). RST. GS8) ;
<NØ2> CD56 := C563/CD56 := GND ;
CD56 représente la commande : 0 → DB2

COMMANDES (TAMP, ADH)
C55 := (DC72.GS3 + DEC1 + DC73 + DC74) ;
<Ø1> C53 ← DB7 ;
C56 := (¬C53. MX1. (¬TIHB)) ;
<Ø2> C58 ←(C55 + C56) ;
<Ø1> DEC ←(PHD3 + C58)

NDEC := ¬DEC ;

<Ø2> C520 ←DC113 ;
<NØ2> CD52 := C520/ CD52 := GND ;
* CD52 représente la commande : 0 → ADH *

<Ø2> C181 ← (GR8 + MEX2) ;
<NØ2> CD18 := C181/CD18 := GND ;
* CD18 représente la commande : TAMP → ADH
<Ø2> C161 ←(¬DC116.GS3) ;
<NØ2> CD16 := C161/CD16 := GND ;
CD16 représente la commande = TAMP→ DB

<Ø2> C151 ← (T2 + GR7) ;
<NØ2> CD15 := C151/CD15 := GND ;
CD15 représente la commande : DB→ TAMP

COMMANDES (S.P)

DS43 := (CAL1. DC124) ;
DSL3 := ¬(DC4.DS43) ;
<Ø2> C221 ← DSL3 ;
<Ø1> C222 ← C221 ;
<Ø2> C223 ← C222 ;
<NØ2> CD22 := C223/CD22 := GND ;
* CD22 représente la commande : DB→ SPL
<Ø2> C170 ← SLD2 ;
<NØ2> CD17 := C170 CD17 := GND ;

* CD17 représente la commande : SPH → DB*
<Ø2> C204 ← (¬DC115.GS1 + (¬DC116. GS2) + DC126. GR7 + GS7) ;
<NØ2> CD20 := C204/CD20 := COND ;

* CD20 représente la commande : AB → SP *

<Ø2> C210 ← DSL3 ;
<NØ2> CD21 := C210/CD21 := GND ;

CD21 représente la commande : DB → SPH
C241 :=(¬SLBC. GSO) ;
<Ø2> C242 ← C241 ;
<NØ2> CD24 := 242/CD24 := GND ;
CD24 représente la commande = SPL → ADL
<Ø2> C243 ← C241 ;
<NØ2> CDH24 := C243/ CDH2← := GND ;
* CDH24 représente la commande : SPH →ADH *
SLD2 := (DC4-DS43) ;
<Ø2> C252 ←SLO2 ;
<Ø1> C253 ←C252 ;
<Ø2> C254 ←C253 ;
<M12> CD25 := C254/ CD25 := GND ;
*CD25 représente la commande : SPL→DB *

```
* COMMANDES (RCC) *

 <Ø2>   PS12  ← (DC21 + GS6) ;
 PSDB  := (PS12.Ø1) ;
 PSO   := (¬RE7F. PSDB) ;

 * PSO représente la commande : C →DBO *

 PS1   := (¬V. PSDB) ;
*PS1 représente la commande : V → DB1*

 PS2   := (¬Z. PSDB) ;
 * PS2 représente la commande : Z → DB2*

 PS3   := (¬N. PSDB) ;
*PS3 représente la commande : N → DB3*
 PS4   := (¬IM. PSDB) ;
*PS4 représente la commande : I → DB4*
 PS5   :=  (¬RE3F. PSDB) ;
*PS5 représente la commande : H →DB5*

*COMMANDES (X) *

 C293  :=  (DC81. CAL1 + GR5) ;
<Ø2>   C294 ← C293 ;
<NØ2>  CD29 :=   C294/CD29 := GND ;

*  CD29 représente la commande DB → XH *

<Ø2>   CD295 ← C293 ;
<Ø1>   C296  ← C295 ;
<Ø2>   C297  ← C296 ;
<NØ2>  CD31  := C297/CD31 := GND ;

*CD31 représente la commande : DB →XL *
<Ø2>   C261 ← SBLC ;
<NØ2>  CD26  := C261/CD26 := GND ;
*CD26 représente la commande : XH → ADH *
```

```
  <Ø1>  C272 ←XHD3 ;
 XHO2   := (CAL1 . DC71) ;
 <Ø2>   C274 ← (XHO2 + DC68. GS3 + C 272) ;
 <NØ2> CD27   := C274/CD27   := GND ;
 *CD27 représente la commande : XH  DB*

<Ø2>    C320 ← XHL2 ;
<Ø1>    C321 ← C320 ;
<Ø1>    C322 ←XHD3 ;
<Ø2>    C325 ← (SLBC + C321 + C322) ;
<NØ2>   CD32 := C325/ CD32 := GND ;
*CD32   représente la commande : XL → BOP*

<Ø2>    C331 ←XHO2 ;
<Ø1>    C332 ←C331 ;
<Ø2>    C334 ←(C332 + DC68. GS2) ;
<NØ2>   CD33   := C334 / CD33 := GND ;
* CD33  représente la commande : XL→ DB*

 XHL2   := (CAL1. DC70) ;
<Ø2>    C280 ← XHL2 ;
<NØ2>   CD28 := C280/CD28 := GND ;
*CDZ8   représente la commande : XH → BOP*

<Ø2>    C305 ← (CAL2. DC83 + DC84-GS1) ;
<NØ2>   CD30   := C305/CD30 := GND ;
*CD30   représente la commande : AD → X*
```

COMMANDES (BOP)

```
<Ø2>    C511 ← (SLBC + MXO + GS3) ;
<NØ2>   CD51   := C511/CD51 := GND ;
 *CD51 représente la commande : ADL → BOP*
```

```
<Ø2>      C532 ← ((DC66 + DC67). CAL1) ;
<NØ2>     CD53 := C532/CD53 := GND ;
*CD53     représente la commande : O → BOP.*
```

COMMANDES (ACCB)

```
 CBL4     :=   (DC61. CAL1) ;
<Ø2>      C360 ← CBL4 ;
<Ø1>      C361 ← C360 ;
 C362  := (DC57 + DC49B) ;
 C365  := (R53 + C361 + CAL2. C362 + GS2. DC58) ;
<Ø2>   C366 ← ((  DC55). ( DC66). C365) ;
<NØ2>  CD36 := C366/CD36 := GNO ;
*CD36  représente la commande : DB →ACCB *

 C351   := ((DC51 + DC52). CAL1) ;
<Ø2>    C355  ←((GS5 + C351 + DC50). DC80) ;
<NØ2>   CD35  := C355/ CD35 := GND ;
  *CD35 représente la commande : ACCB → DB*

<Ø2>      C341 ←   ((¬ DC59). (¬DC66). CBL4) ;
<NØ2>     CD34     := C341/CD34 := GND ;
*CD34  représente la commande : ACCB → BOP*
```

COMMANDES (ACCA)

```
 LAC1  :=  ((DC46 + DC47). DC45. CAL1) ;
<Ø2>      C374  ← LAC1 ;
<Ø1>      C375 ←  C374 ;
 C376  :=((DC384 + DC42). CAL2 + C375) ;
<Ø2>      C370  ←(GE4 + (¬DC55. C376) + DC48.GS2) ;
<NØ2>     CD37 := C370 / CD37 := GND ;
*CD37 représente la commande : DB→ ACCA*
```

```
 C381  :=      ((DC39 + DC383 + DC37). CAL1) ;
<Ø2>   C384  ←(DC40.GS4 + DC41 + C381)
<NØ2>  CD38   :=  C384/CD38 := GND ;
```

CD38 représente la commande : ACCA→ DB

```
<Ø2>   C391 ← (¬DC44. LAC1) ;
<NØ2>  CD39  := C391/CD39 := GND ;
```

CD39 représente la commande : ACCA → BOP

COMMANDES (UAL)

```
 DAA2  :=      (DC9. CAL1);
<Ø2>   C600 ←  DAA2 ;
<NØ2>  CD60 := C600/CD60 := GND ;
```

*CD60 représente la commande : COST →ADD *

```
 C612  := (DB7.DC4 + (¬DC4). RE7F) ;
<NØ2>  CD61 := C612/CD61 := GND ;
```

CD61 représente la commande : R7E (bit entrant décalage)

```
<Ø2>   C631  ← ¬(DAA2 + DBB3 + DBB2) ;
<NØ2>  CD63 := C631/CD63 := GND ;
```

CD63 représente la commande : DB → ADD

```
 DBB3  := ((DC33 + DC34). (¬MX0)) ;
<Ø2>   C621 ← DBB3 ;
<NØ2>  CD62 :=  C621/CD62 := GND ;
```

CD62 représente la commande : NDB → ADD

```
<Ø2>   C640 ← DBB3 ;
<NØ2>  CD64  := C640/ CD64 := GND ;
```

CD64 représente la commande : RD (décalage droit)

```
 SLOI    :=    ( ¬DC23.CAL2 + MXI) ;
<Ø1>     C65I    SLOI ;
<NØ2>    CD65 := C65I/CD65 := GND ;
*CD65 représente la commande : T.VAL → DB*

 AND2    :=   ( ¬(DC24+DC25) + MXO + DC26) ;
<Ø2>     C66I  ←ET2 ;
 C664    :=      (¬DCI7.DC3I.RE7F + ( ¬REF7F). DC3O + DC28 + DC29) ;
<Ø2>     C666 ← ( ¬MXO. C644) ;
<Ø1>     C668 ← (C66I + C666) ;
<NØ2>    CD66 := C668/CD66 := GND ;
*CD66 représente la commande : CIN*

<Ø2>     C67I ←(¬MXO. DC22) ;
<Ø1>     C672 ← C67I ;
 CD67    := C672 ;
* CD67 représente la commande : DG (décalage gauche)*

<Ø2>     C68O ←ET2 ;
<Ø1>     C68I ← C68O ;
 CD68    := C68I ;
*CD68 représente la commande

<Ø2>     C69O  ←OU2 ;
<Ø1>     OU  ← (C69O.DCI) ;
 CD69    := ¬ OU ;
*CD69 représente la commande : NON (complément OU)*
```

```
 OU2    :=      (DC35.CAL1) ;
<Ø2>    C700 ← OU2 ;
<Ø1>    OU1  ← C700 ;

 CD70   :=   OU1 ;
 * CD70 représente la commande : complément OU1*
```

COMMANDES (T.I.N.)

```
 C450   :=  ¬(SL01 + GR8 + WRT4 + DC18 + GS3) ;
<Ø2>    C451 ← (GR7 + C450 + GS3) ;
<NØ2>   CD45 = C451/ CD45 := GND ;
 CD45 représente la commande : TIN→ DB
```

COMMANDES (T.OUT)

```
<NØ2>   CD46 := DBRW/CD46 := GND ;
*CD46 représente la commande DB →TOUT*
```

ANNEXE,2 - CODES INVALIDES DES MICROPROCESSEURS 6800

Les caractéristiques particulières dans l'exécution de certains codes invalides ayant été exposées, nous présentons dans ce qui suit les résultats obtenus à partir de l'étude approfondie de l'architecture interne des microprocesseurs 6800.

Cette étude a été réalisée en plusieurs étapes et à des niveaux différents:

- études des caractérsitiques du jeu d'instructions,
- étude des masques du circuit intégré,
- simulation sur un schéma logique complet,
- simulation sur une carte d'application 6800.

Les résultats de la simulation des codes invalides sont présentés en détail, aussi bien au niveau de l'architecture interne qu e celui du programmeur, en termes de transferts.

1. - Séquencement général

Pendant l'exécution d'une instruction, le microprocesseur communique avec son environnement essentiellement par son bus de données externe D, son bus d'adresse A, sa sortie de lecture/écriture R/W et sa sortie de validation du bus adresse VMA.

Aussi, nous présentons, dans ce qui suit, le séquencement des actions élémentaires, aussi bien au niveau de la partie opérative qu'au niveau des échanges du microprocesseur avec l'environnement extérieur (diagramme des temps).

Les descriptions sont réalisées pour chacune des étapes de l'exécution d'une instruction:

- séquence Acquisition (avec les différents cas),
- séquence Adressage (avec les différents modes d'adressage).

Pour une question d'encombrement, la description de la séquence opération n'est pas présentée dans cette annexe. Cependant le calcul du séquencement des opérations pour l'ensemble des instructions existe dans [29] qui est plus détaillé du point de vue technique en ce qui concerne l'étude des microprocesseurs 6800.

1.1. - Séquence ACQUISITION

La demande d'acquisition d'une instruction est contrôlée par le bloc [ACQUISITION] du séquenceur suivant les conditions définies par l'organigramme général de fonctionnement.

Le séquencement pendant le premier cycle dépend de l'instruction exécutée précédemment (instruction de branchement ou non). Cependant, ce cycle concerne toujours le chargement du code opération de l'instruction suivante dans le registre instruction RI. Il existe trois cas différents de séquencement dont la différence essentielle est la source de l'adresse de l'instruction suivante.

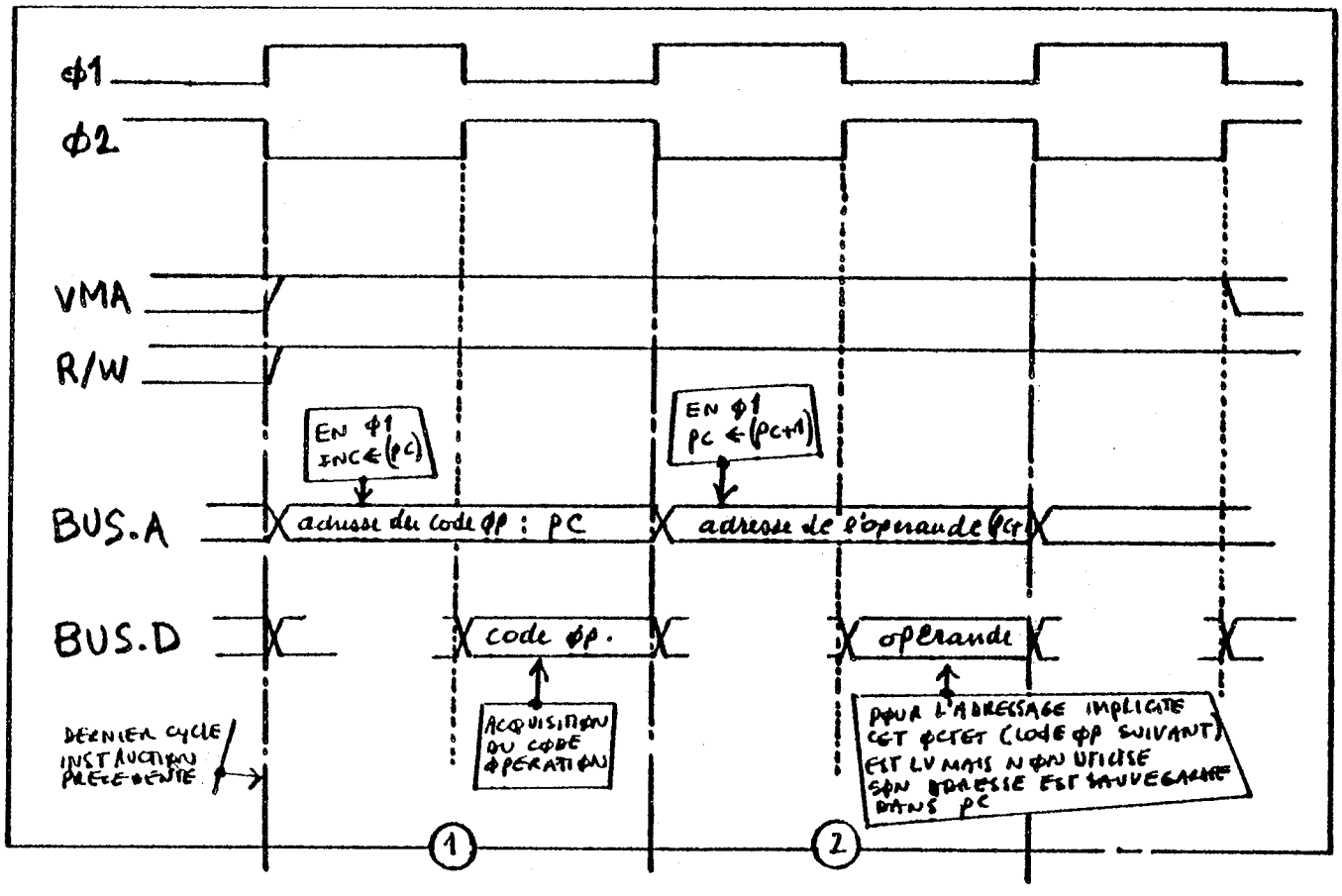


FIG. 68 - Diagramme des temps de la séquence ACQUISITION.

Nous présentons dans ce qui suit le séquencement de ces trois cas :

1er CAS :

L'instruction précédente n'est pas celle d'un branchement (ou une instruction de branchement dont le test est faux).

Cycle (1)

phase $\phi 1$: PCL → ABL → INCL(+1), PCH → ADH → INCH ;
Bus.A = (PC) adresse du code opération

phase $\phi 2$: INCL → T.INCL , INCH → T.INCH ,
Bus.D (contenant le code opération) → RI.

Cycle (2)

phase $\phi 1$: T.INCL → ADL & PCL, ADL → INCL(+1),
T.INCH → ADH & PCH, ADH → INCH ,
Bus.A = (PC+1) adresse de l'opérande
/ à ce niveau on a : PC ← (PC+1) /

phase $\phi 2$: INCL → T.INCL, INCH → T.INCH ,
Bus.D (contenant l'opérande) → T.IN.

Fin de la séquence ACQUISITION pour le cas 1.

2ème CAS ;

L'instruction précédente est une instruction de branchement dont le test est vrai, ou une instruction avec adressage immédiat.

Cycle (1)

phase $\phi 1$: T.INCL → ADL → INCL(+1), T.INCH → ADH → INCH ,
Bus.A = @(adresse de branchement calculée par l'instruction précédente).

phase $\phi 2$: INCL → T.INCL, INCH → T.INCH,
Bus.D (contenant le code opération) → RI

Cycle (2)

phase $\phi 1$: T.INCL → ADL & PCL, ADL → INCL(+1),
T.INCH → ADH & PCH, ADH → INCH,
Bus.A = (@+1) adresse de l'opérande
/ à ce niveau on a : PC ← (@+1) /

phase $\phi 2$: INCL → T.INCL, INCH → T.INCH ,
Bus.D (contenant l'opérande) → T.IN.

Fin de la séquence ACQUISITION du cas 2.

3ème CAS

Une interruption (NMI, IRQ, RESET) ou un branchement, est en traitement.
Notons qu'au début de cette séquence d'acquisition, l'octet de poids fort (H) de l'adresse du code opération a été chargé dans le tampon TAMP. Le tampon d'entrée T.IN contient l'octet de poids faible (L) de cette adresse.

Cycle (1)

phase $\phi 1$: T.IN(=L) → Bus.DB → INCL(+1),
TAMP → ADH → INCH,
Bus.A = @ (@=HL) adresse du code opération de la prochaine instruction.

phase $\phi 2$: INCL → T.INCL, INCH → T.INCH,
Bus.D (contenant le code opération) → RI.

Cycle (2)

phase $\phi 1$: T.INCL → ADL& PCL, ADL → INCL(+1),
T.INCH → ADH & PCH, ADH → INCH,
Bus.A = (@+1) adresse de l'opérande

phase $\phi 2$: INCL → T.INCL, INCH → T.INCH,
Bus.D (contenant l'opérande) → T.IN.

Fin de la séquence ACQUISITION pour le cas 3.

1.2. - Séquence adressage

Adressage implicite / Adressage accumulateur

Dans ces modes d'adressage, on n'a pas de séquence adressage distincte (durée nulle). La séquence opération succède immédiatement à la séquence acquisition. L'instruction dans ces modes d'adressage est formée d'un seul octet.

La séquence acquisition (dans ces modes d'adressage) fait apparaître le code opération de l'instruction suivante au cycle(2). Ce code opération sera perdu dans le tampon d'entrée T.IN.

Ce cycle sera recommencé au début de la séquence acquisition de l'ins-

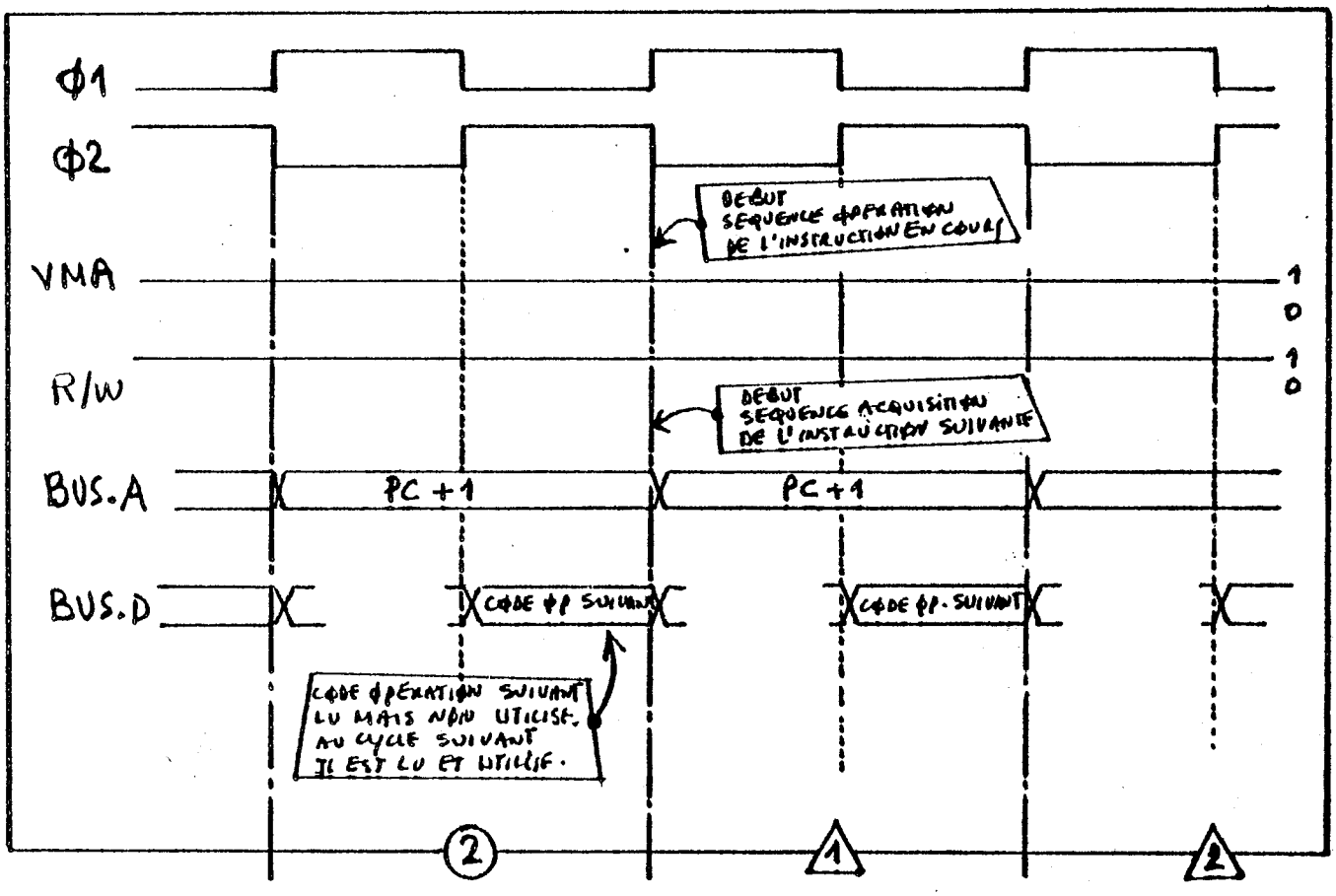


FIG. 69 - Diagramme des temps pour les modes d'adressage IMPLICITE/ACCUMULTEUR

Adressage immédiat.

Cas général

- durée en cycles d'horloge = 0
- nombre d'octets = 2

Dans ce mode d'adressage, le premier octet de l'instruction contient le code opération et le deuxième octet contient l'opérande. Il n'existe pas de séquence d'adressage distincte, dès la fin de la séquence acquisition, la séquence opération commence. Le cycle (3) de la figure (70) n'existe que pour les instruction CPX, LDX, LDS.

Cas particulier des instructions CPX, LDX, LDS

- durée en cycles d'horloge = 1
- nombre d'octets = 3

Le deuxième octet = poids fort , troisième octet = poids faible.

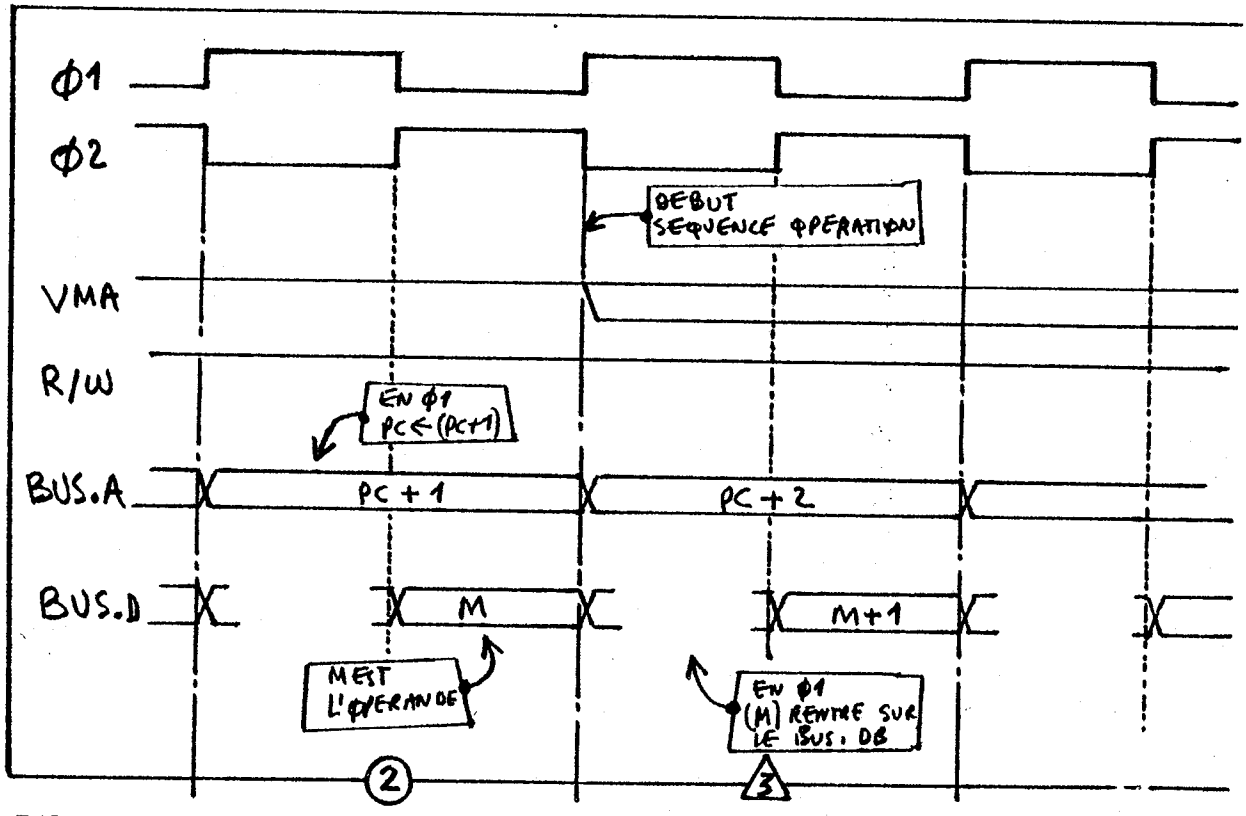


FIG. 70 - Diagramme des temps pour le mode d'adressage IMMEDIAT.

Adressage indexé

La durée d'exécution de cette séquence d'adressage est de trois cycles d'horloge.
Dans ce mode d'adressage, les instructions possèdent deux octets. Le premier octet étant le code opération, le deuxième représente le déplacement (d) qu'il faut ajouter à la valeur de l'index pour obtenir l'adresse de l'opérande. Le contenu du registre index n'est pas modifié.

Le cycle (5) de la figure (71) est différent pour les instructions STA, STS, STX, car VMA=0 alors que pour les autres instructions on a: VMA=1.

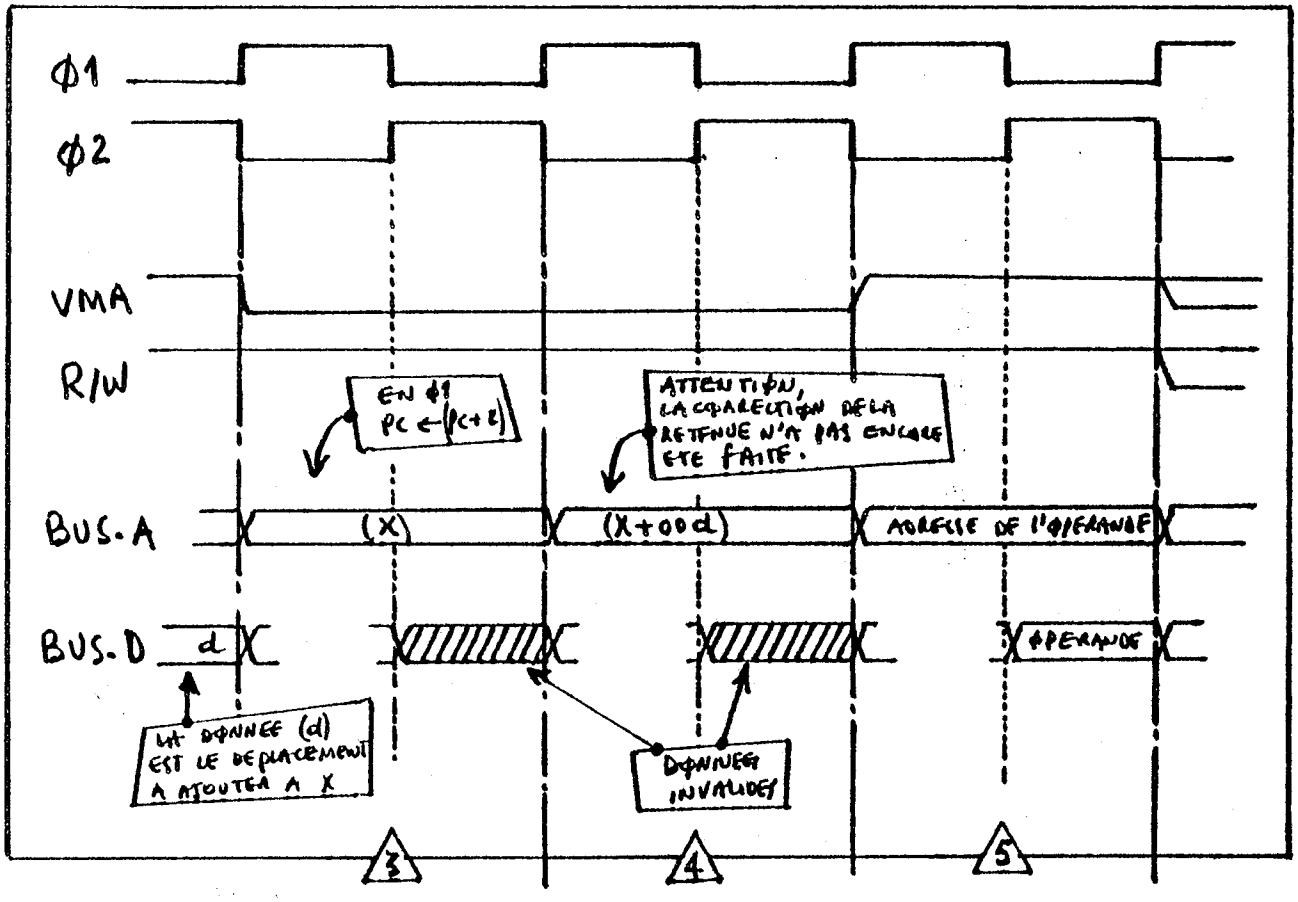


FIG. 71 - Diagramme des temps pour le mode d'adressage INDEXE.

Séquencement

Cycle (3) - début de la séquence adressage

phase ϕ1 : T.IN(=d) → BUS.DB → TAMP,
XL → BUS.ϕP → ADL → INCL(+0), XH → ADH → INCH,
BUS.DB(=d) & BUS.ϕP(=XL) → UAL(+,CIN=0),
T.INCL → PCL, T.INCH → PCH / à ce niveau on a = PC ← (PC+2) /

phase ϕ2 : la retenue C7 générée par l'opération (XL+d) rentre dans la partie contrôle qui calcule la commande de INCH:
(+0) si C7=0, (+1) si C7=1,
INCL → T.INCL, INCH → T.INCH, UAL → T.UAL,
La donnée sur le BUS.D (non utilisée) est invalide car VMA=0.

Cycle (4)

phase ϕ1 : T.UAL → BUS.DB → INCL(+0)
T.INCH → ADH → INCH(+C7) /correction de la retenue pour les poids forts/

phase ϕ2 : INCL → T.INCL, INCH → T.INCH,
la donnée sur le BUS.D est invalide car VMA=0.

Cycle (5) - Cas général des instructions VMA=1

phase ϕ1 : T.INCL → ADL → INCL(0), T.INCH → ADH → INCH,
BUS.A=@ (@=X+d) adresse de l'opérande

phase ϕ2 : INCL → T.INCL, INCH → T.INCH,
BUS.D (contenant l'opérande (M)) → T.IN

Cycle (5) - Cas particulier des instructions STA, STS, STX : VMA=0

phase ϕ1 : T.INCL → ADL → INCL(+0), T.INCH → ADH → INCH
BUS.A=@ (@=X+d) adresse de l'opérande.

phase ϕ2 : INCL → T.INCL, INCH → T.INCH,
BUS.D (contenant une donnée invalide) → T.IN
/ cette donnée invalide sera perdue dans T.IN /

Fin de la séquence adressage INDEXE.

Adressage relatif

La durée d'exécution de cette séquence d'adressage est de deux cycles d'horloge.

Dans ce mode d'adressage, les instructions possèdent deux octets. Ce mode d'adressage est réservé aux instructions de branchement.
Le premier octet étant le code opération, le deuxième octet représente un déplacement (d) qui est interprété par le microprocesseur comme un nombre binaire en complément à deux (le bit de poids fort indique le signe). Il est possible de se brancher à une adresse mémoire située dans la gamme (-128), (+127) par rapport à l'adresse de l'instruction suivante.

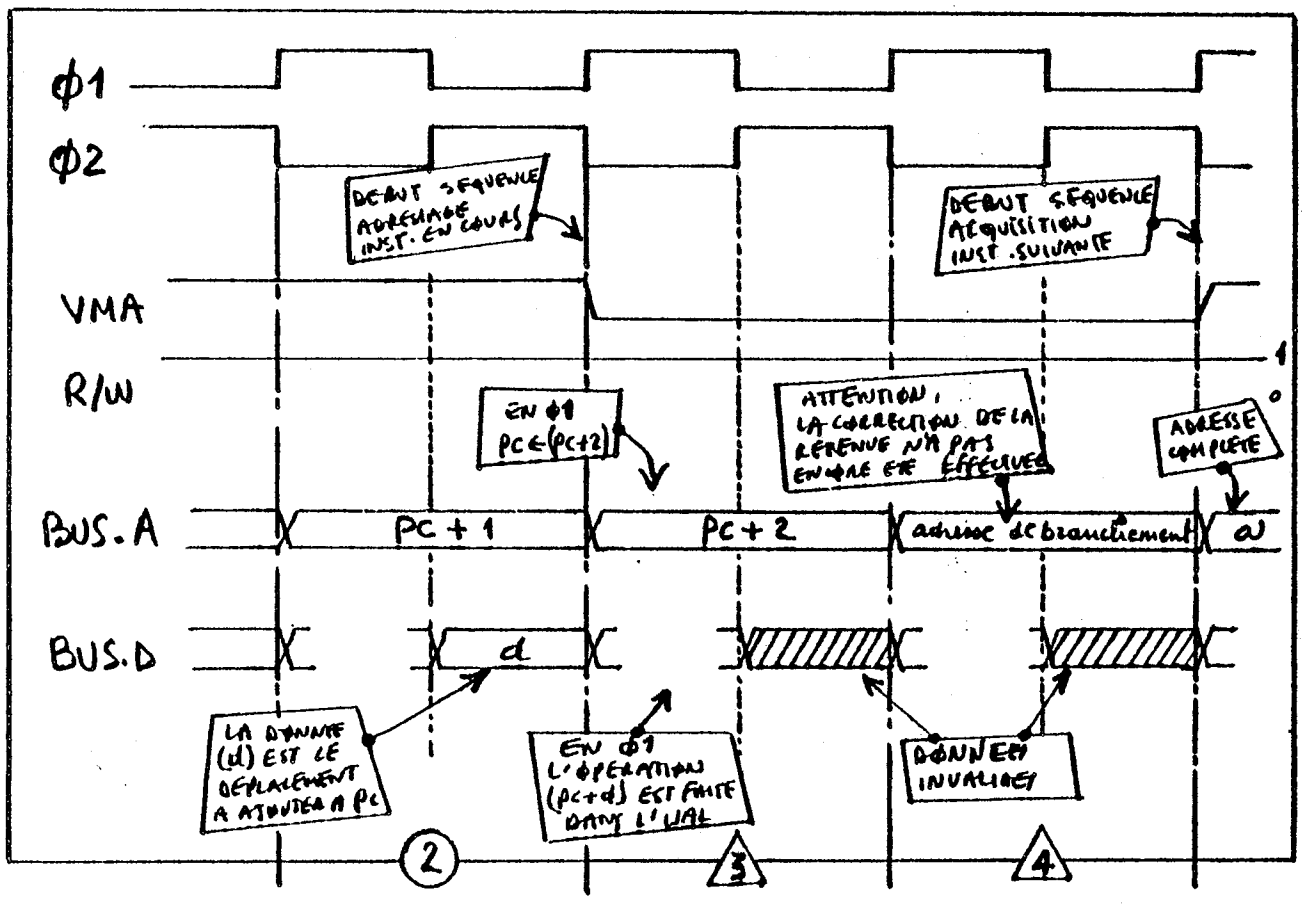


FIG. 72 - Diagramme des temps pour le mode adressage RELATIF.

Séquencement

Cycle (3) - début de la séquence adressage, VMA=0

phase ϕ1 : T.IN(contenant le déplacement(d))→ BUS.DB → TAMP,
T.INCL(nouveau PCL) → ADL → BUS.ϕP, ADL → INCL(+0)
T.INCH(nouveau PCH) → ADH → INCH
BUS.DB(=d) & BUS.ϕP(=PCL) UAL('+'),
/ DB7 : bit du signe, est échantilloné par la partie contrôle /

phase ϕ2 : INCL → T.INCL, INCH → T.INCH,
/ la retenue C7 générée par l'opération (PCL+d) est
échantillonnée par la partie contrôle /,
BUS.D (contenant une donnée invalide) → T.IN
/ Cette donnée invalide sera perdue dans T.IN /

Cycle (4) - correction de l'octet poids fort, VMA=0

phase ϕ1 : T.UAL → BUS.DB → INCL(+0),
T.INCH → ADH → INCH(CH).
La commande (CH) de l'incrémenteur INCH est calculée par la
partie contrôle en fonction de C7 et DB7 comme suit:
si DB7=1 et C7=0 alors (CH)=(-1)
si DB7=1 et C7=1 alors (CH)=(+0)
si DB7=0 et C7=0 alors (CH)=(+0)
si DB7=0 et C7=1 alors (CH)=(+1)

phase ϕ2 : INCL → T.INCL, INCH → T.INCH,
/ l'adresse de branchement est prête, elle est dans T.INC /
BUS.D (contenant une donnée invalide) → T.IN,
/ cette donnée invalide sera perdue dans T.IN /

ADRESSAGE DIRECT

La durée d'exécution de cette séquence d'adressage est d'un cycle d'horloge. Dans ce mode d'adressage, les instructions possèdent deux octets. Le deuxième octet de l'instruction représente l'adresse de l'opérande (Les bits d'adresse de poids fort sont forçés à zero).

Séquencement

Cycle (3) . Début de la séquence adressage, cas général, VHA = 1.

Phase ∅1 : T.IN (= à poids faible de l'adresse)→ BUS.DB → TAMP,
BUS.DB →INCL (+0), "OO"→ ADH →INCH,
BUS.A =@(@ = OOa) adresse de l'opérande
T.INCL →PCL, T.INCH →PCH/à ce niveau : PC← (PC + 2)/

Phase ∅2 : INCH → T.INCL, INCH → T.INCH,
BUS. D (contenant l'opérande)→ T.IN.

Cycle (3) Début de la séquence adressage pour les instructions ;
STA, STX, ST5 : VMA = 0.

Phase ∅1 : T.IN (= a)→BUS. DB → TAMP,
BUS.DB →INCL (+ 0), "OO"→ ADH →INCH
BUS.A =@(@= OOa) mais la donnée est invalide car VMA = 0

Phase ∅2 : INCL →T.INCL → INCH → T.INCH
BUS.D (contenant la donnée invalide) → T.IN.

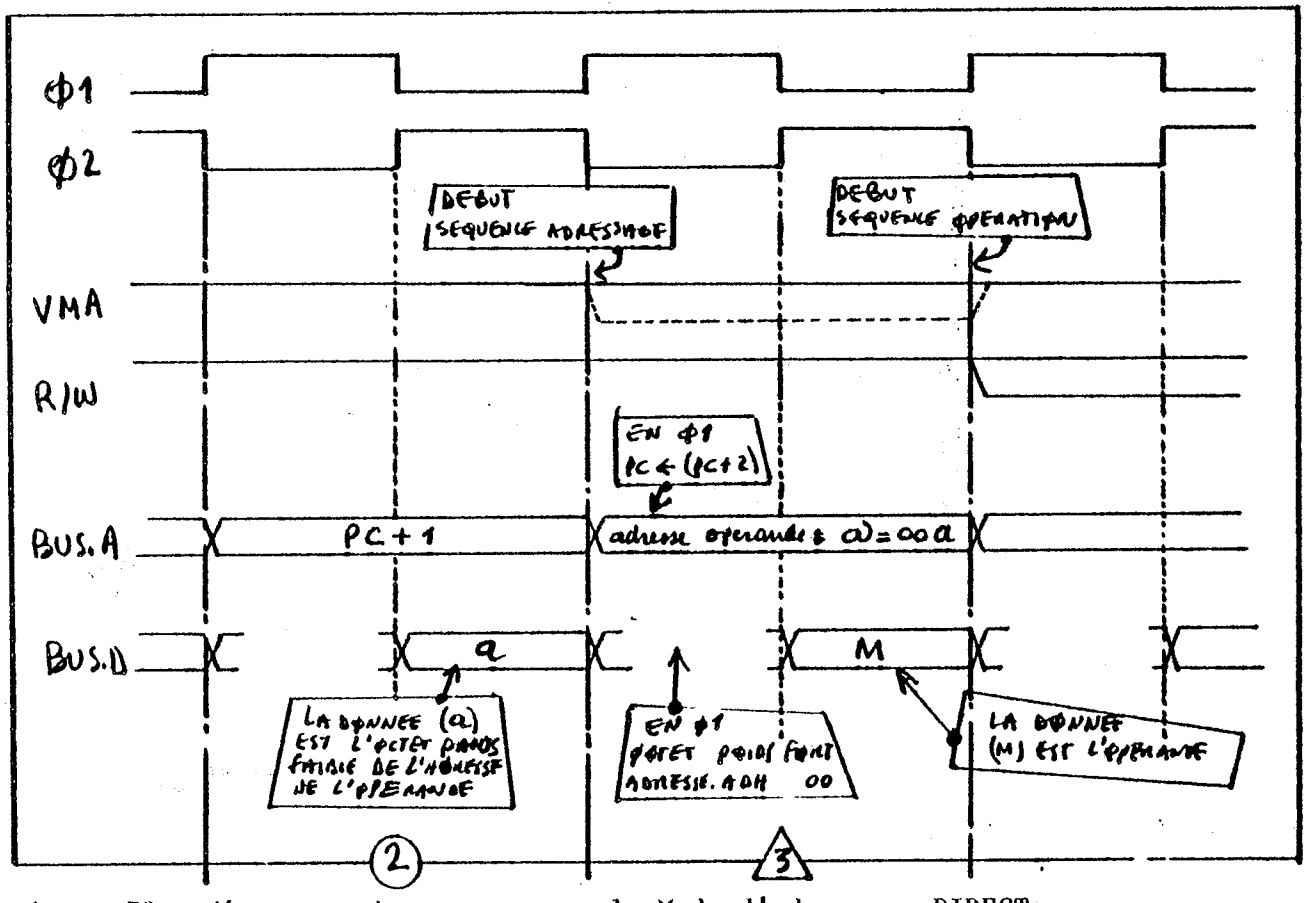


Figure 73 : diagramme des temps pour le Mode d'adressage DIRECT

ADRESSAGE ETENDU

La durée d'exécution de cette séquence d'adressage est de deux cycles d'horloge.
Dans ce mode d'adressage, les instructions possèdent trois octets.
L'adresse mémoire de l'opérande est complètement définie par les deux octets qui suivent le code opération.

Le deuxième octet est le poids fort de l'adresse.
Le troisième octet est le poids faible de l'adresse.

Le cycle (4) est différent pour les instructions STA, STS, STX car VMA = 0, alors que pour les autres instructions VMA = 1.

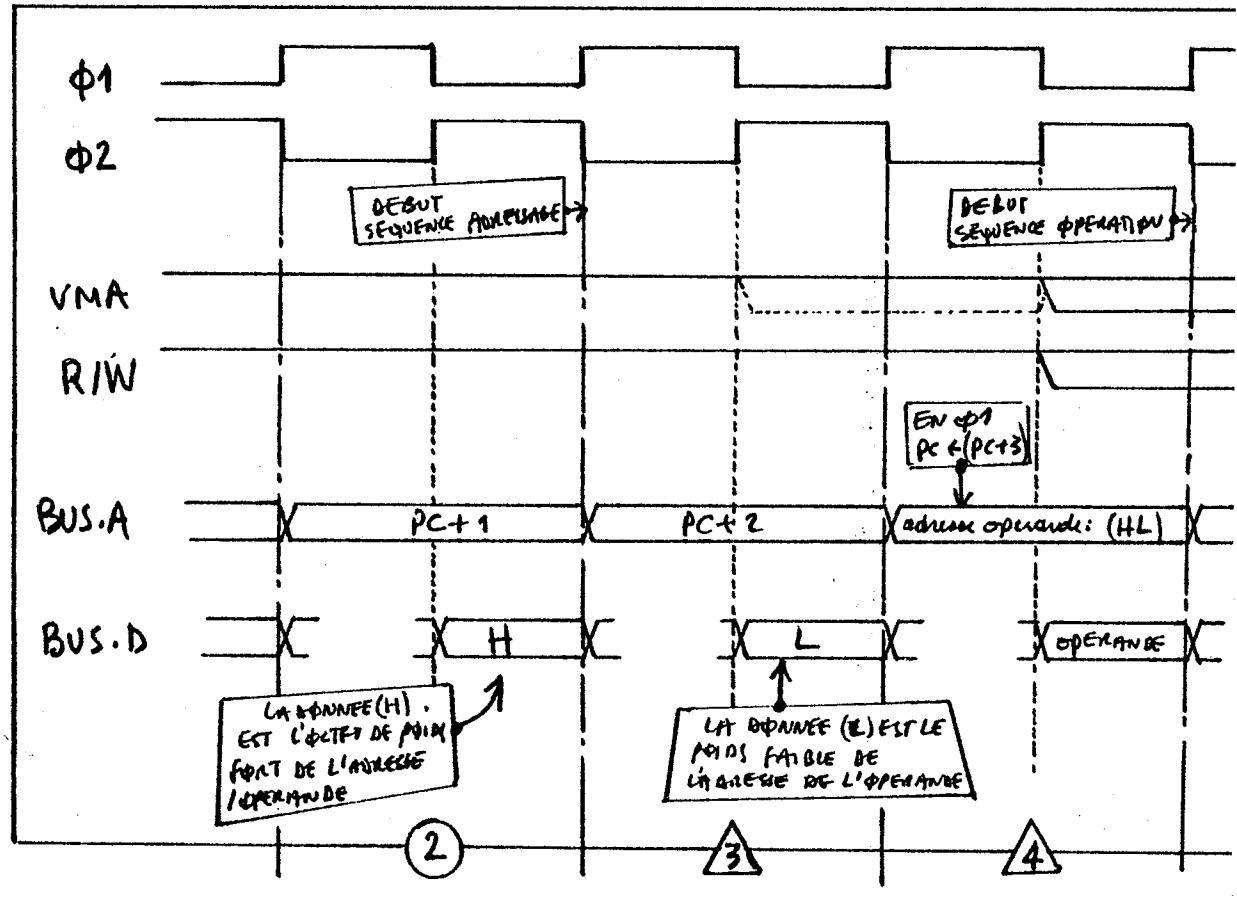


Fig. 74 : Diagramme des temps pour le mode d'adressage ETENDU

SEQUENCEMENT

Cycle (3) Début de la séquence adressage

Phase Ø1 : T.IN (= H, poids fort de l'adresse) → BUS. DB → TAMP,
T.INCL → (ADL → INCL (+1), T.INCH → ADH → INCH,
BUS.A = (PC +2) adresse de l'octet poids faible

Phase Ø2 : INCL → T.INCL, INCH → T.INCH,
BUS.D (contenant poids faible (L) de l'adresse) → T.IN

Cycle (4) Cas général des instructions : VMA = 1

Phase Ø1 : T.IN (=L, poids faible de l'adresse) → BUS.DB,
BUS.DB → INCL(.) TEMP (= H) → ADH → INCH
BUS.A = (HL) adresse de l'opérande.
T.INCL → PCL, TINCH → PCH/ à ce niveau : PC ← (PC + 3)/

Phase Ø2 : INCL → TINCL, INCH → T.INCH,
BUS.D (contenant l'opérande) → T.IN.

Cycle (4) : Cas particulier des instructions STA, STB, STX : VMA = 0

Phase Ø1 : T.IN. (=L, poids faible de l'adresse) → BUS. DB,
BUS.DB → INCL (+0), TEMP (= H) → ADH → INCH,
BUS.A = (HL) adresse de l'opérande
T.INCL → PCL, T.INCH → PCH/à ce niveau : pc ←(PC + 3)/

Phase Ø2 : INCL → T.INCL , INCH → T.INCH
BUS.D (contenant une donnée invalide) → T.IN
/Cette donnée invalide sera perdue dans T.IN/

2.- TRAITEMENT DES CODES INVALIDES

2.1. Caractéristiques générales :

L'éxécution d'un code invalide provoque toujours les mêmes actions. Ces actions sont essentiellement de trois types :

(1) Modification des registres internes (écriture) ,
(2) Modification d'une zone mémoire (écriture) ;
(3) Bouclage dans l'exécution (lecture sequentielle et sans arrêt de l'espace mémoire).

D'un certain point de vue, on peut considérer que le jeu d'instruction du 6800 s'enrichi par l'utilisation de certain des codes invalides dans les programmes.

Après l'exécution d'un code invalide, à l'exception des codes invalides (3C, 3D, 9D, DD, 87, 8F, C7, CF), l'instruction suivante est chargée à l'adresse normale qui est fonction du Mode d'adressage du code invalide.

L'utilisation des codes invalides dans un programme, en tenant compte des caractéristiques particulières (notamment celles du registre des codes conditions RCC), ne pose aucun problème. Il est même conseillé de le faire pour certains codes invalides intéressants comme par exemple 14 , 15, 3A, ...

Les codes invalides du 6800 constituent, par leur nombre, un moyen efficace pour la détection des pannes. Les parties contrôles (décodage et interpretation du code opération) du 6800 et du 6802 sont les mêmes par conséquent leurs codes invalides sont identiques.

Nous presentons dans ce qui suit les operations realisées par chacun des 59 codes invalides.

CODES	OPERATIONS - REALISEES	ADRESSAGE		MODIF. (RCC)					
		~	#	5 H	4 I	3 N	2 Z	1 V	0 C
00	Equivalent à NOP	IMPLICITE	2 1	•	•	•	•	•	•
02	Equivalent à NOP	IMPLICITE	2 1	•	•	•	•	•	•
03	Equivalent à NOP	IMPLICITE	2 1	•	•	•	•	•	•
04	Equivalent à NOP	IMPLICITE	2 1	•	•	•	•	•	•
05	Equivalent à NOP	IMPLICITE	2 1	•	•	•	•	•	•
12	$A+\overline{B} \longrightarrow A$	IMPLICITE	2 1	•	•	↑	↑	↑	↓
13	$A+\overline{B} \longrightarrow A$	IMPLICITE	2 1	•	•	↑	↑	↑	↓
14	$A \wedge B \longrightarrow A$	IMPLICITE	2 1	•	•	↑	↑	R	•
15	Test : $A \wedge B$	IMPLICITE	2 1	•	•	↑	↑	R	•
18	Equivalent à DAA	IMPLICITE	2 1	•	•	↑	↑	↑	•
1A	$A+B \longrightarrow A$	IMPLICITE	2 1	↑	•	↑	↑	↑	•
1C	$A+B+1 \longrightarrow A$	IMPLICITE	2 1	•	•	↑	↑	↑	•
1D	$A+B \longrightarrow A$	IMPLICITE	2 1	•	•	↑	↑	↑	↑
1E	Equivalent à TBA	IMPLICITE	2 1	•	•	↑	↑	R	•
1F	Equivalent à TBA	IMPLICITE	2 1	•	•	↑	↑	R	S
21	Equivalent à BCS	RELATIF	4 2	•	•	•	•	•	•

$(A \wedge B)$: produit logique

$(A + B)$: addition arithmétique

CODES	OPERATIONS - REALISEES	ADRESSAGES		MODIF. (RCC)					
		~	#	5 H	4 I	3 N	2 Z	1 V	
38	Equivalent à RTS	IMPLICITE	5	1	•	•	•	•	•	
3A	Equivalent à RTI	IMPLICITE	10	1	•	•	•	•	•	
3C	BOUCLAGE ... (**)	IMPLICITE	∞	∞	•	•	•	•	•	
3D	BOUCLAGE ... (**)	IMPLICITE	∞	∞	•	•	•	•	•	
41	Modification du Rcc en fonction de A	IMPLICITE	2	1	•	•	④	↑	②	
42	$\overline{A}$ ⟶ A	IMPLICITE	2	1	•	•	↑	↑	R	
45	Modification du Rcc en fonction de A	IMPLICITE	2	1	•	•	R	④	⑤	
4B	A-1 ⟶ A	IMPLICITE	2	1	•	•	↑	↑	↑	
4E	Equivalent à NOP	IMPLICITE	2	1	•	•	•	•	•	
51	Modification du Rcc en fonction de B	IMPLICITE	2	1	•	•	④	↑	②	
52	$\overline{B}$ ⟶ B	IMPLICITE	2	1	•	•	↑	↑	R	
5B	B-1 ⟶ B	IMPLICITE	2	1	•	•	↑	↑	↑	
55	Modification du Rcc en fonction de B	IMPLICITE	2	1	•	•	R	④	⑤	
5E	Equivalent à NOP	IMPLICITE	2	1	•	•	•	•	•	

CODES	OPERATIONS - REALISEES	ADRESSAGES	~	*	MODIF. (Rcc)	5 H	4 I	3 N	2 Z	1 V	0 C
61	Equivalent à NEG	INDEXE	7	2		•	•	↑	↑	↑	↓
62	Equivalent à COM	INDEXE	7	2		•	•	↑	↑	R	S
65	Equivalent à LSR	INDEXE	7	2		•	•	R	↑	⑦	↑
6B	Equivalent à DEC	INDEXE	7	2		•	•	↑	↑	R	↑
71	Equivalent à NEG	ETENDU	6	3		•	•	↑	↑	↑	↓
72	Equivalent à COM	ETENDU	6	3		•	•	↑	↑	R	S
75	Equivalent à LSR	ETENDU	6	3		•	•	R	↑	⑦	↑
7B	Equivalent à DEC	ETENDU	6	3		•	•	↑	↑	R	↑
83	$A + \bar{M} \longrightarrow A$	IMMEDIAT	2	2		•	•	↑	↑	↑	↓
87	$A \longrightarrow M_{(PC+2)}$	IMMEDIAT	3	3		•	•	↑	↑	R	•
8F	$SP_H \longrightarrow M_{(PC+2)}$; $SP_L \longrightarrow M_{(PC+3)}$	IMMEDIAT	4	4		•	•	⑧	↑	R	•
93	$A + \bar{M} \longrightarrow A$	DIRECT	3	2		•	•	↑	↑	↑	↓
9D	BOUCLAGE ... (*)	DIRECT	∞	∞		•	•	•	•	•	•
A3	$A + \bar{M} \longrightarrow A$	INDEXE	5	2		•	•	↑	↑	↑	↓
B3	$A + \bar{M} \longrightarrow A$	ETENDU	4	3		•	•	↑	↑	↑	↓

CODES	OPERATIONS - REALISEES	ADRESSAGES		MODIF. (RCC)					
		~	#	5 H	4 I	3 N	2 Z	1 V	0 C
C3	$B + \bar{M} \longrightarrow B$	IMMEDIAT	2 2	•	•	↑	↑	↑	↓
C7	$B \longrightarrow M_{(PC+2)}$	IMMEDIAT	3 3	•	•	↑	↑	R	•
CC	Equivalent à CPX	IMMEDIAT	3 3	•	•	(9)	↑	(10)	•
CD	Equivalent à BSR	IMMEDIAT	8 2	•	•	•	•	•	•
CF	$X_H \longrightarrow M_{(PC+2)}$; $X_L \longrightarrow M_{(PC+3)}$	IMMEDIAT	4 4	•	•	(8)	↑	R	•
D3	$B + \bar{M} \longrightarrow B$	DIRECT	3 2	•	•	↑	↑	↑	↓
DC	Equivalent à CPX	DIRECT	4 2	•	•	(9)	↑	(10)	•
DD	BOUCLAGE ... (*)	DIRECT	∞ ∞	•	•	•	•	•	•
E3	$B + \bar{M} \longrightarrow B$	INDEXE	5 2	•	•	↑	↑	↑	↓
EC	Equivalent à CPX	INDEXE	6 2	•	•	(9)	↑	(10)	•
ED	Equivalent à JSR	INDEXE	8 2	•	•	•	•	•	•
F3	$B + \bar{M} \longrightarrow B$	ETENDU	4 3	•	•	↑	↑	↑	↓
FC	Equivalent à CPX	ETENDU	5 3	•	•	(9)	↑	(10)	•
FD	Equivalent à JSR	ETENDU	9 3	•	•	•	•	•	•

2.2. Codes invalides du type - I

La totalité des codes invalides à l'exception des codes (3C, 3D, DD, 87, C7, CF), appartiennent au type I.

Si le code invalide a été exécuté par erreur, alors le type-I, entraîne des perturbations au niveau des registres internes. Ceci peut entraîner une propogation d'erreurs tout au long des opérations qui suivent cette exécution. Si par contre le code invalide a été utilisé exprès par le programmeur, alors le déroulement du programme s'effectue tout à fait normalement et l'utilisation d'un code invalide de type (I) dans un programme ne pose aucun problème.

Cependant, il faut faire attention à l'opération qui est réalisée par ce code invalide. par exemple :

- Le code 14 réalise l'opération : AND (A,B) → A
  RCC modifié correctement
- Le code 1A réalise l'opération : (A + B) → A
  le bit (C) du RCC n'est pas modifié
- Le code 3A quant à lui est équivalent partiellement à l'instruction RTI car tous les registres sont restaurés à partir de la pile correctement, sauf le registre code condition RCC qui n'est pas modifié *

Le diagramme des temps de chaque code invalide du type-I est le même que celui de l'instruction équivalente.

---

* Ce code invalide est très intéressant car sa programmation permet le positionnement du RCC par interruption.

2.3. Codes invalides du type - 2 :

Les codes invalides qui appartiennent au type-2 sont 87, 8F, C7, CF.

Si le code invalide a été exécuté par erreur alors le type-2 entraîne des perturbations au niveau de la mémoire (écriture en mémoire). De plus on risque d'avoir une exécution en chaîne des codes invalides. Ceci peut-être observé si l'instruction qui suit l'un de ces codes invalides, possède un adressage étendu (opérande pris pour code opération et réciproquement).

Ces quatres codes invalides sont équivalents à l'instruction STORE avec une différence quant au nombre d'octet nécessaire à son exécution.

Code invalide 87 :

- la durée d'exécution est de 3 cycles d'horloge,
- il est équivalent à l'instruction STAA qui est composée de trois octets :
  Octet 1 = code opération
  Octet 2 = opérande (non utilisé)
  Octet 3 = emplacement de stockage de ACCA.

Il en est de même pour le code invalide C7 qui est équivalent à STAB.

Code invalide 8F :

- La durée d'exécution est de 4 cycles d'horloge,
- Il est équivalent à l'instruction STS qui est composée de quatre octets :

Octet 1 = code opération
Octet 2 = opérande (non utilisé),
Octet 3 = emplacement pour stocker SPH,
Octet 4 = emplacement pour stocker SPL.

Il en est de même pour le code invalide CF qui est équivalent à STX.

Si par contre le code invalide a été utilisé exprés dans le programme, alors son exécution ne perturbe pas le déroulement du programme.

On peut considérer que ces codes déterminent de nouvelles instructions STAA , STAB, STS, STX avec un mode d'adressage particulier : pseudo-immédiat (rangement dans le format de l'instruction).

Les diagrammes de temps et le séquencement de ces codes invalides, nous donnent tous les détails de ces opérations.

SEQUENCEMENT DU CODE INVALIDE 87 POUR ACCA (RESP. C7 POUR ACCB) :

Cycle (1) : début de la séquence acquisition

Phase Ø1 : PCL → ADL → INCL (+1), PCH → ABH → INCH,
BUS.A = (PC) adresse du code opération (invalide).

Phase Ø2 : INCL → T.INCL, INCH → T.INCH,
BUS.D (contenant le code opération) → R.I.
/Le décodeur est déconnecté de la partie contrôle et du registre instruction RI/.

Cycle (2) :

Phase Ø1 : T.INCL → ADL PCL & ADL → INCL (+ 1)
T.INCH → ADH PCH, & ADH → INCH,
BUS.A = (PC + 1) adresse de l'opérande M
/à ce niveau on a : PC ← (PC + 1)/

Phase Ø2 : INCL → T.INC , INCH → T.INCH,
le BUS.D contient l'opérande (M) qui sera perdue.

Cycle (3) :

Phase Ø1 : T.INCL → ADL & PCL, ADL → INCL (+ 1)
T.INCH → ADH & PCH, ADH → INCH,
BUS.R = (PC + 2) / à ce niveau on a : pc ← (PC + 2)/
ACCA → BUS.DB → T.OUT,
BUS.DB (= ACCA) & BUS.OP (=FF)→ UAL ('^')
/le test dans l'UAL sert à corriger le RCC/

Phase Ø2 : INCL → T.INCL, INCH →T.INCH,
T.OUT → BUS.D/ la valeur de ACCA rentre dans M (PC + 2)/.

Cycle (1) : Début de la séquence acquisition de l'instruction suivante :

Phase Ø1 : le RCC est corrigé,
T.INCL → ADL →INCL (+1), T.INCH → ADH → INCH,
BUS.A = (PC + 3) adresse du code opération de l'instruction suivante.

SEQUENCEMENT DU CODE INVALIDE 8F pour S (RESP CF pour X) :

Cycle (1). : Début de la séquence Acquisition

Phase ∅1 : PCL → ADL → INCL (+1), PCH → ADH → INCH,
BUS.A = (PC) adresse du code opération (invalide)

Phase ∅2 : INCL → T.INCL, INCH → T.INCH
BUS.D (contenant le code opération) → RI
/le décodeur est déconnecté de la partie contrôle et du registre instruction/.

Cycle (2) :

Phase ∅1 : T.INCL → ADL & PCL, ADL → INCL (+1),
T.INCH → ADH & PCH, ADH → INCH,
BUS.A = (PC + 1) adresse de l'opérande M
/à ce niveau on a : PC ← (PC + 1)/

Phase ∅2 : INCL → T.INCL, INCH → T.INCH,
le BUS.D contient l'opérande (M) qui sera perdue

Cycle (3) :

Phase ∅1 : T.INCL → ADL & PCL, ADL → INCL (+1),
T.INCH → ADH & PCH, ADH → INCH,
BUS.A = (PC +2)/à ce niveau on a : PC ← (PC + 2)/.
SPH → BUS.DB → T.OUT, BUS-DB BUS.OP (=FF) → UAL ('^')
/l'Opération AND dans l'UAL sert à corriger le RCC/.

Phase ∅2 : INCL → T.INCL, INCH → T.INCH,
T.OUT → BUS.D/la valeur de SPH rentre dans M (PC + 2)/.

Cycle (4) :

Phase ϕ1 : Le RCC est corrigé partiellement/correction bit Z non achevée/
T.INCL → ADL → INCL (+1), T.INCH → ADH → INCH,
BUS.A = (PC + 3), SPL→ BUS.DB → T.OUT,
BUS.DB & BUS. OP (= FF) → UAL ('^')
/l'opération AND dans l'UAL sert à corriger le RCC/.

Phase ϕ2 : INCL → T.INCL, INCH →T.INCH,
T.OUT BUS.D/la valeur de SPL rentre dans M (PCK)/.

Cycle (1) . Début de la séquence acquisition de l'instruction suivante.

Phase 01 : Le RCC est complétement corrigé
T.INCL → ADL → INCL (+1), T. INCH → ADH → INCH
BUS.A = (PC + 4) adresse du code opération de l'instruction suivante.

FIG. 76

DIAGRAMME DES TEMPS DU CODE INVALIDE 87 POUR ACCA (Resp.C7 pour ACCB)

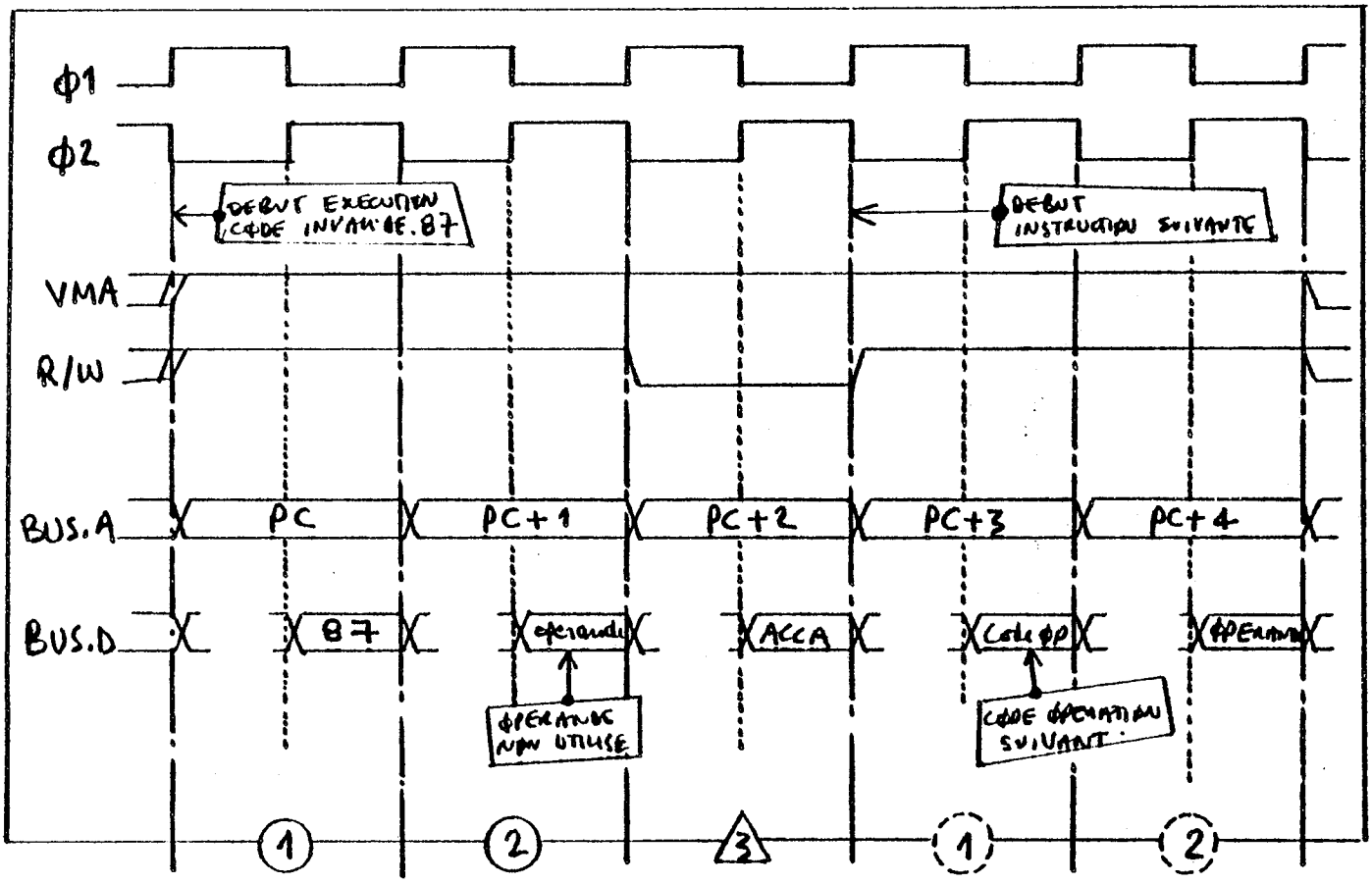
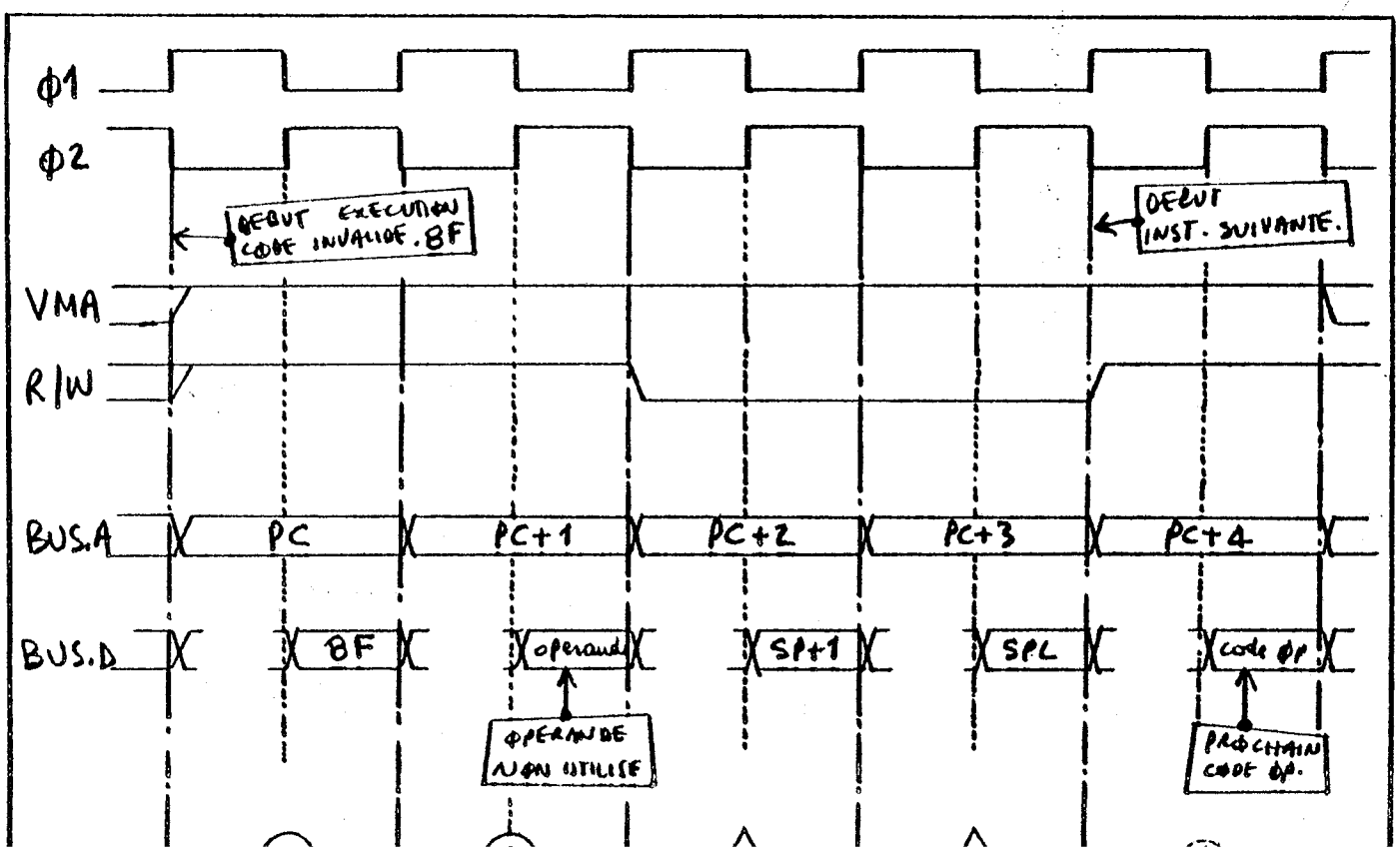


FIG. 77

DIAGRAMME DES TEMPS DU CODE INVALIDE 8F POUR S (RESP CF POUR X)



2.4. Codes invalides du type -3 :

Les codes invalides qui appartiennent au type -3 sont :

3C, 3D, 9D, DD

Ces codes invalides réalisent un bouclage dans l'exécution, c'est à dire une lecture sequentielle et sans arrêt de l'espace mémoire.

L'exécution provoquéepar l'un de ces quatre codes ne se termine jamais. Aussi, les différentes interruptions n'ont aucun effet sur le microprocesseur. La seule solution pour sortir de ce blocage est d'envoyer le signal d'initialisation [$\overline{\text{RESET}}$ = 0] .

Ceci est dû à une caractéristique du 6800, les interruptions ne sont traitées qu'à la fin de l'instruction en cours.

Le type -3 ne modifie rien, mais peut poser de graves problèmes, en particulier pour les programmes des applications "en temps réel" qui utilisent des informations externes (interruptions).

On peut assembler ces quatres codes invalides en deux groupes :

Le premier groupe : 3C, 3D.

- Mode adressage implicite.
- Ces deux codes invalides sont assimilés au départ respectivement aux instructions WAI (3E) et SWI (3F) dont l'exécution commence par une séquence de sauvegarde. Donc, l'exécution de ces deux codes entraîne la sauvegarde dans la pile de PCH et PCL (PC contient l'adresse du code opération suivant).

  Ensuite, l'exécution se résume à une lecture sequentielle et sans arrêt de tout l'espace mémoire à partir de l'adresse (SP-2)/Tous les blocs du séquenceur sont désactives/
  La valeur qui est lue en mémoire rentre dans l'UAL, mais le résultat n'est pas utilisé.

Les registres RCC, X, A, B ne sont pas modifiés et PC ←(PC + 1), SP ← (SP-1)/ par la suite PC et SP ne sont plus modifiés/.

Le deuxième groupe : 9D, DD.

- mode adressage direct
- La première consiste à chercher l'opérande à l'adresse 00 a (a : octect de poids faible de l'adresse). L'opérande rentre sur le bus DB mais n'est pas utilisé.

L'octet de poids faible de l'adresse est sauvegarde dans le tampon TAMP. Ensuite, l'exécution se résume en une lecture séquentielle et sans arrêt de tout l'espace mémoire à partir de l'adresse (a + 1)/ Tous les blocs du séquenceur sont désactivés/.

La valeur lue en mémoire rentre dans l'UAL, mais le résultat n'est pas utilisé.

Les registres RCC, X, SP, A, B ne sont pas modifiés et PC← (PC + 2)/par la suite PC n'est plus modifié.

SEQUENCEMENT DES CODES INVALIDES 3C, 3D :

Cycle (1). : Début de la séquence acquisition

Phase ϕ1 : PCL→ ADL→ INCL (+1), PCH →ADH →INCH,
BUS.A = (PC) adresse du code opération (invalide)

Phase ϕ2 : INCL → T.DNCL, INCH → T.INCH,
BUS.D (contenant le code opération) → RI.

Cycle (2).

Phase ϕ1 : T.INCL → ADL & PCL, ADL → INCL (+1),
T.INCH → ADH & PCH, ADH → INCH,
BUS.A = (PC + 1) adresse de la pseudo-opérande (M).
/à ce niveau on a : PC← (PC + 1)/

Phase ϕ2 : INCL → T.INCL, INCH → T.INCH,
BUS.D contient l'opérande (M) qui sera perdue

Cycle (3).

Début de la séquence opération

Phase ϕ1 : SPL → ADL →INCL (-1), SPH →ADH → INCH,
BUS.A = (SP) première adresse de sauvegarde,
PCL → BUS. DB → T.OUT.

Phase ϕ2 : INCL → T.INCL, INCH → T.INCH,
T.OUT→ BUS.D/la valeur de PCL rentre dans M (SP)/.

Cycle (4) .

Phase Ø1 : T.INCL → ADL → INCL (-1) & SPL,
T.INCH → ADH → INCH & SPH
/à ce niveau on a : SP ← (SP-1)/,
BUS.A = (SP-1) deuxième adresse de sauvegarde,
PCH → BUS.DB → T.OUT.

Phase Ø2 : INCL → T.INCL, INCH →T.INCH,
T.OUT → BUS.D/la valeur de PCH rentre dans M (SP-1)/.

Cycle (5).

Phase Ø1 : T.INCL → ADL → INCL (+1), T.INCH →ADH → INCH,
BUS.A = (SP-2) adresse du premier octet lu en mémoire,

Phase Ø2 : INCL → T.INCL, INCH → T.INCH,
BUS.D (contenant l'octet M (SP-2)) → T.IN.

Cycle (6).

Phase Ø1 : T.IN (= (M-1)) → BUS.DB
BUS.DB & BUS.OP (=FF) →UAL ('^')
/Le résultat du test de M (SP-2) sera perdu, RCC non modifié/,
T.INCL → ADL →INCL (-1), T.INCH → ADH → INCH,
BUS.A = (SP-1) adresse du deuxième octet lu en mémoire.

Phase Ø2 : INCL → T.INCL, INCH → T. INCH,
BUS.D (contenant l'octet M (SP-1) → T.IN

Cycle (7).

Phase Ø1 : T.IN(= (M-2)) → BUS.DB,
BUS.DB & BUS.OP (=FF) → UAL ('^')
/le résultat du test de M (SP-1) sera perdu, RCC non modifié/
T.INCL → ADL → INCL (+1), T.INCH → ADH → INCH,
BUS.A = (SP-2)

Phase Ø2 : INCL → INCL, →INCH →T.INCH,
BUS.D (contenant l'octet M (SP-2)) → T.IN

Cycle (8).

Phase Ø1 : T.IN (= (M-1)) → BUS.DB
BUS.DB & BUS.OP (=FF) → UAL ('^')
/le résultat du test de M (SP-2) sera perdu, RCC non modifié/
T.INCL → ADL → INCL (+1), T.INCH → ADH → INCH,
BUS.A = (SP-1)

Phase Ø2 : INCL → T.INCL, INCH → T.INCH,
BUS.D (contenant l'octet M (SP-1)) → T.IN

Cycle (9).

Phase Ø1 : T.IN (= (M)) → BUS.DB,
BUS.DB & BUS.OP (=FF) UAL ('^')
/le résultat du test de M(SP-1) sera perdu, RCC non modifié/
T.INCL → ADL→ INCL (+1), T.INCH → ADH → INCH,
BUS.A = (SP).

Phase Ø2 : INCL →T.INCL, INCH →T.INCH.
BUS.D (contenant l'octet M (SP)) → T.IN

Cycle (10)

Cycle (11)

Phase Ø1 : T.IN (= (M+N)) → BUS. DB
BUS.DB & BUS.OP (=FF) → UAL ('^')
/le résultat du test de M (SP + n-1) sera perdu, RCC non modifié/
T.INCL → ADL → INCL (+1), T.INCH → ADH → INCH,
BU.A = (SP + n)

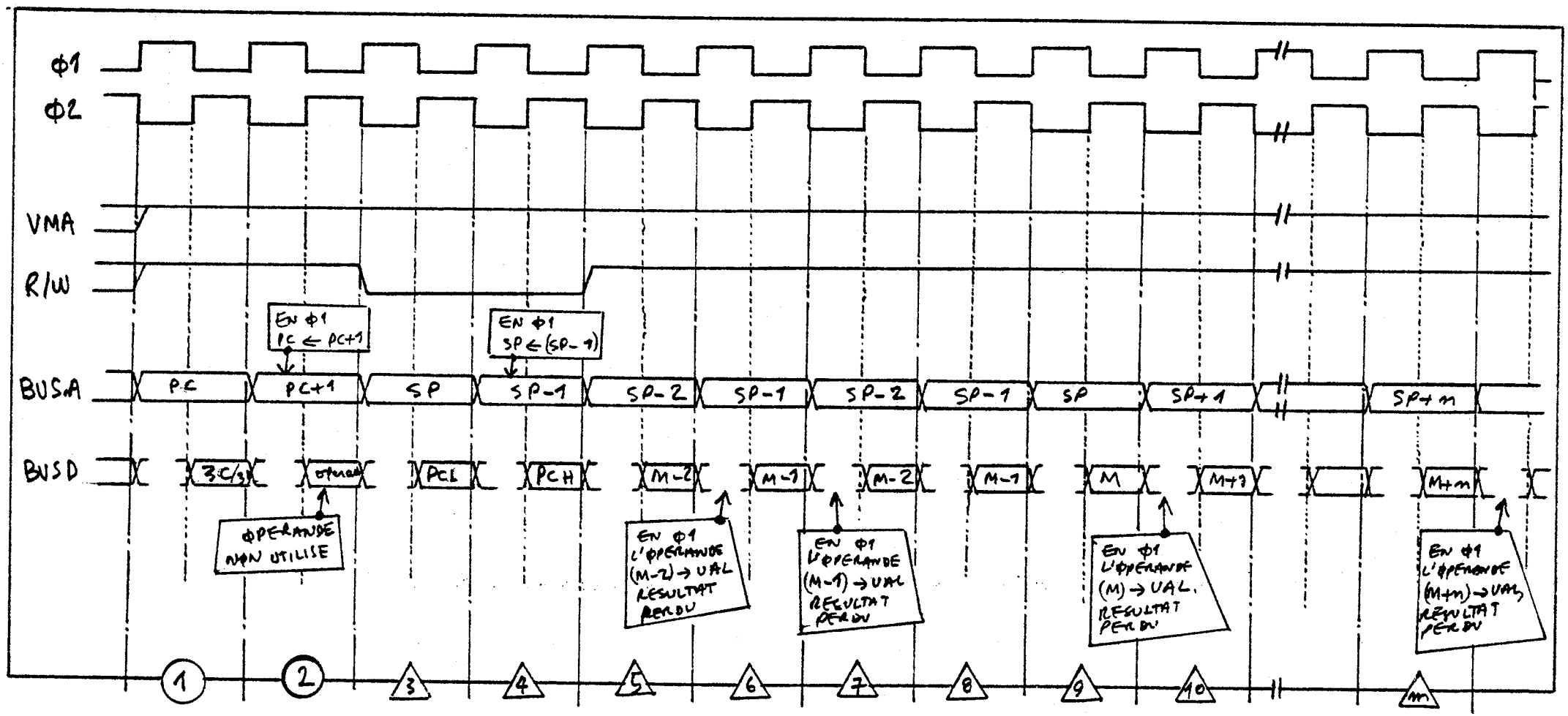
Phase Ø2 : INCL → INCL, INCH → T.INCH,
BUS.D (contenant l'octet M (SP + n)) → T.IN

Cycle (m + 1)

Phase Ø1 : T.IN (= (M + n + 1)) → BUS.DB,
BUS.DB & BUS.OP (= FF) → UAL ('^')
/Le résultat du test de M (SP + n) sera perdu, RCC non modifié
T.INCL → ADL → INCL (+1), T.INCH → ADH → INCH,
BUS.A = (SP + n + 1)

Phase Ø2 :

FIGURE 78 : DIAGRAMME DES TEMPS POUR LES CODES INVALIDES 3C, 3D.



SEQUENCEMENT DES CODES INVALIDES 9D, DD :

Cycle (1). Début de la séquence Acquisition

Phase ϕ1 : PCL → ADL → INCL (+1), PCH →ADH → INCH,
BUS.A = (PC) adresse du code opération (invalide)

Phase ϕ2 : INCL → T.INCL, INCH →T.INCH,
BUS.D (contenant le code opération) → RI.

Cycle (2).

Phase ϕ1 : T.INCL → ADL → PCL, & ADL → INCL (+1),
T.INCH → ADH & PCH, ADH →INCH,
BUS.A = (PC + 1)

Phase ϕ2 : INCL → T.INCL, INCH →T.INCH,
BUS.D (contenant l'octet poids faible (a) de l'adresse de l'opérande) → T.IN.

Cycle (3) Début de la séquence adressage

Phase ϕ1 : T.IN (= a) →BUS.DB →INCL (+ 1) & TAMP
'OO' → ADH → INCH,
BUS.A = ⓐ (ⓐ = ooa) adresse de l'opérande,
T.INCL →PCL, T.INCH → PCH
/à ce niveau on a : PC ← (PC + 2)/

Phase ϕ2 : INCL → T.INCL, INCH → T.INCH
BUS.D (contenant l'opérande (M)) → T.IN

Cycle (4). Début de la séquence opération.

Phase ϕ1 : T.IN (=(M))→ BUS.DB
BUS.DB & BUS.OP (=FF) → UAL (' ^')
/le test de l'octet (M) sera perdu, RCC non modifié/
T.INCL→ ADL → INCL (+1), T.INCH → ADH → INCH,
BUS.A = (@ +1)

Phase ϕ2 : INCL→ T.INCL, INCH → I.INCH,
BUS.D (contenant l'opérande (M + 1)→ T.IN

Cycle (5).

Phase ϕ1 : T.IN (= (M+1)) → BUS.DB,
BUS.DB & BUS.OP (=FF) → UAL ('^')
/le test de l'octet (M+1) sera perdu, RCC non modifié/
T.INCL → ADL → INCL (+1), T.INCH → ADH → INCH,
BUS.A = (@ + 2).

Phase ϕ2 : INCL → T.INCL, INCH → T.INCH,
BUS.D (contenant l'opérande (M + 2)) → T.IN

Cycle (6).

Cycle (M).

Phase ø1 : T.IN (= (m + n)) → BUS.DB
BUS. DB & BUS.OP (=FF) → UAL ('^')
/le test de l'octet (M + n) sera perdu RCC non modifié/,
T.INCL → ADL → INCL (+1), T.INCH → ADH →INCH
BUS.A = (ⓐ + n + 1)

Phase ø2 : INCL → T.INCL, INCH → T.INCH
BUS.D (contenant l'opérande (M + n + 1)) → T.IN

Cycle (m + 1).

Phase ø1 : T.IN (= (M + n + 1)) → BUS. DB,
BUS.DB BUS.OP (= FF) → DAL ('^')
/le test de l'octet (m + n + 1) sera perdu, RCC non modifié/,
T.INCL → ADL → INCL (+1), TINCH → ADH → INCH,
BUS.A = (ⓐ + n + 2)

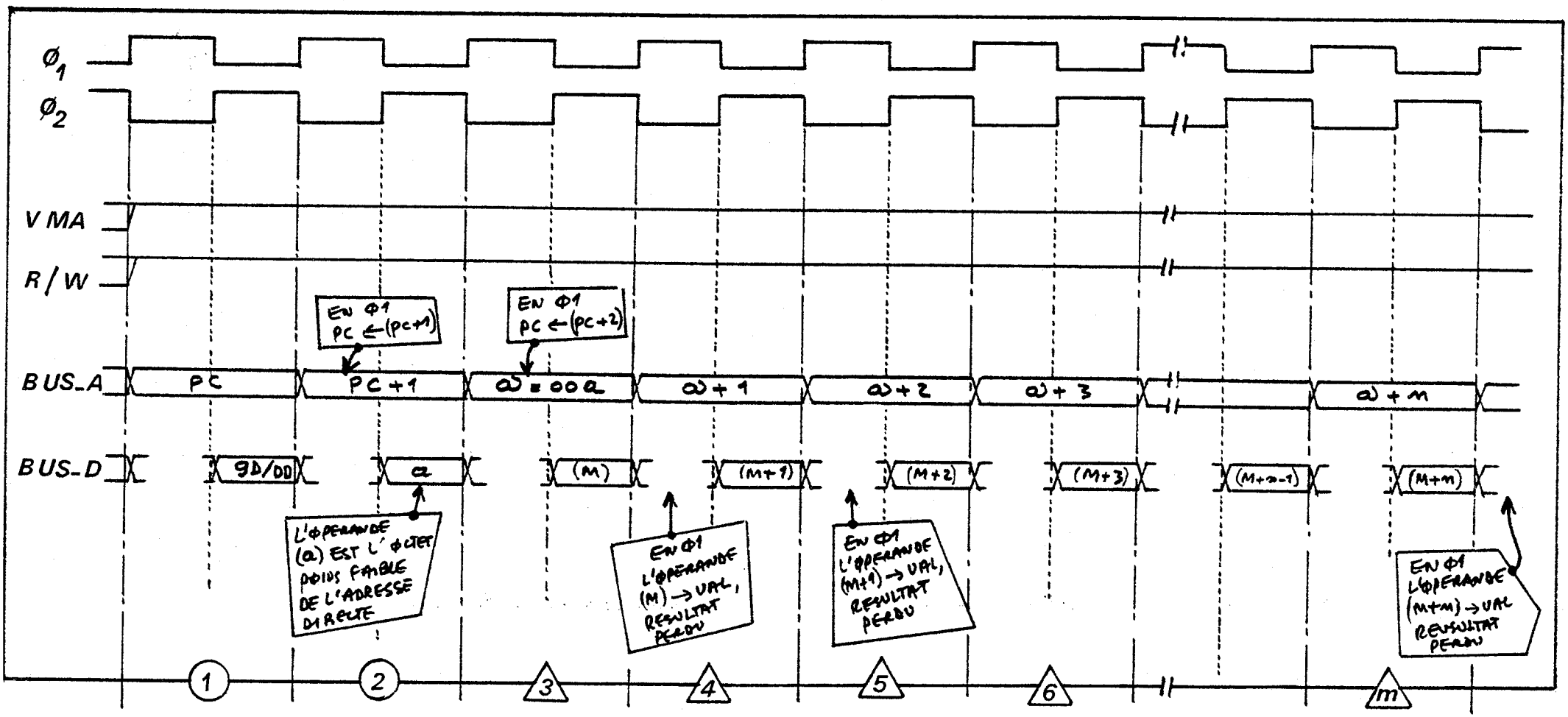
Phase ø2 :

.

.

.

FIG. 79 : DIAGRAMME DES TEMPS POUR LES CODES INVALIDES 9D, DD. .288.



3. Montage externe pour supprimer les anomalies 1, 2 du 6800

FIG.80:

Les masques externes des interruptions NMI et IRQ peuvent etre realisés separement.

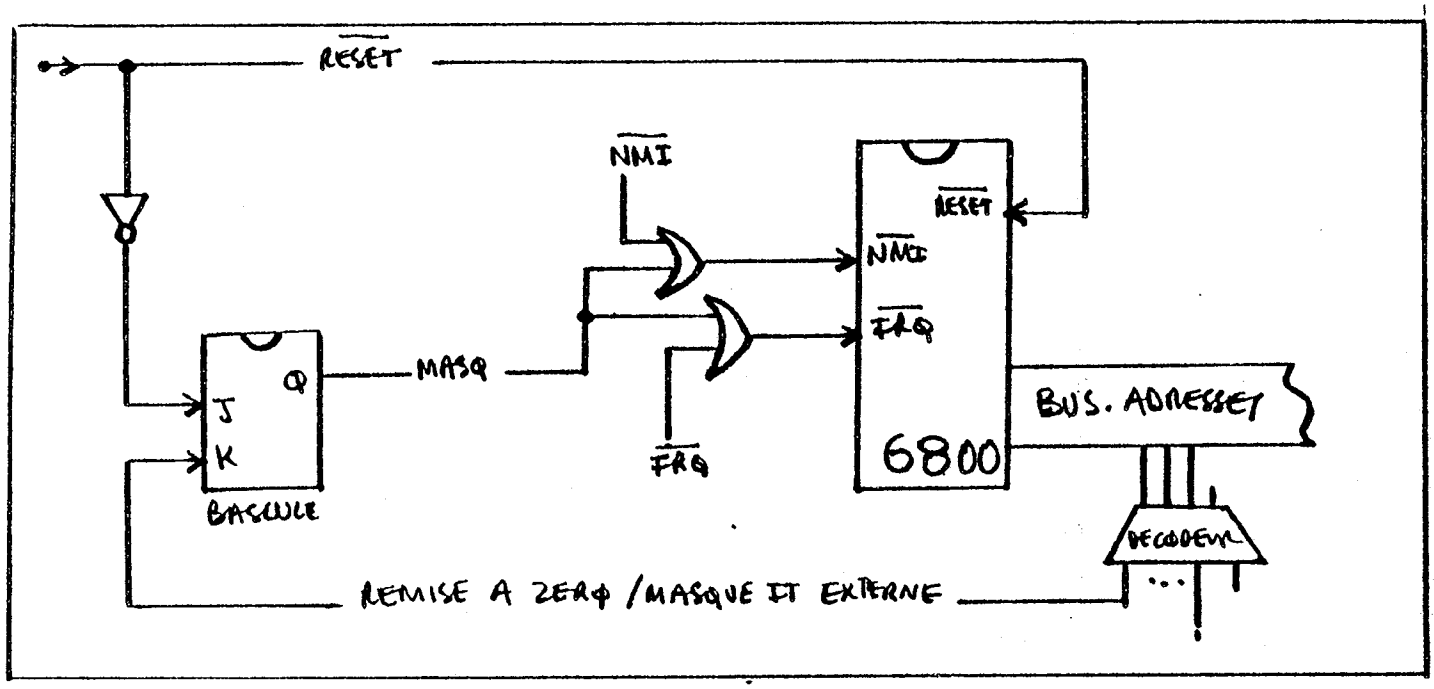
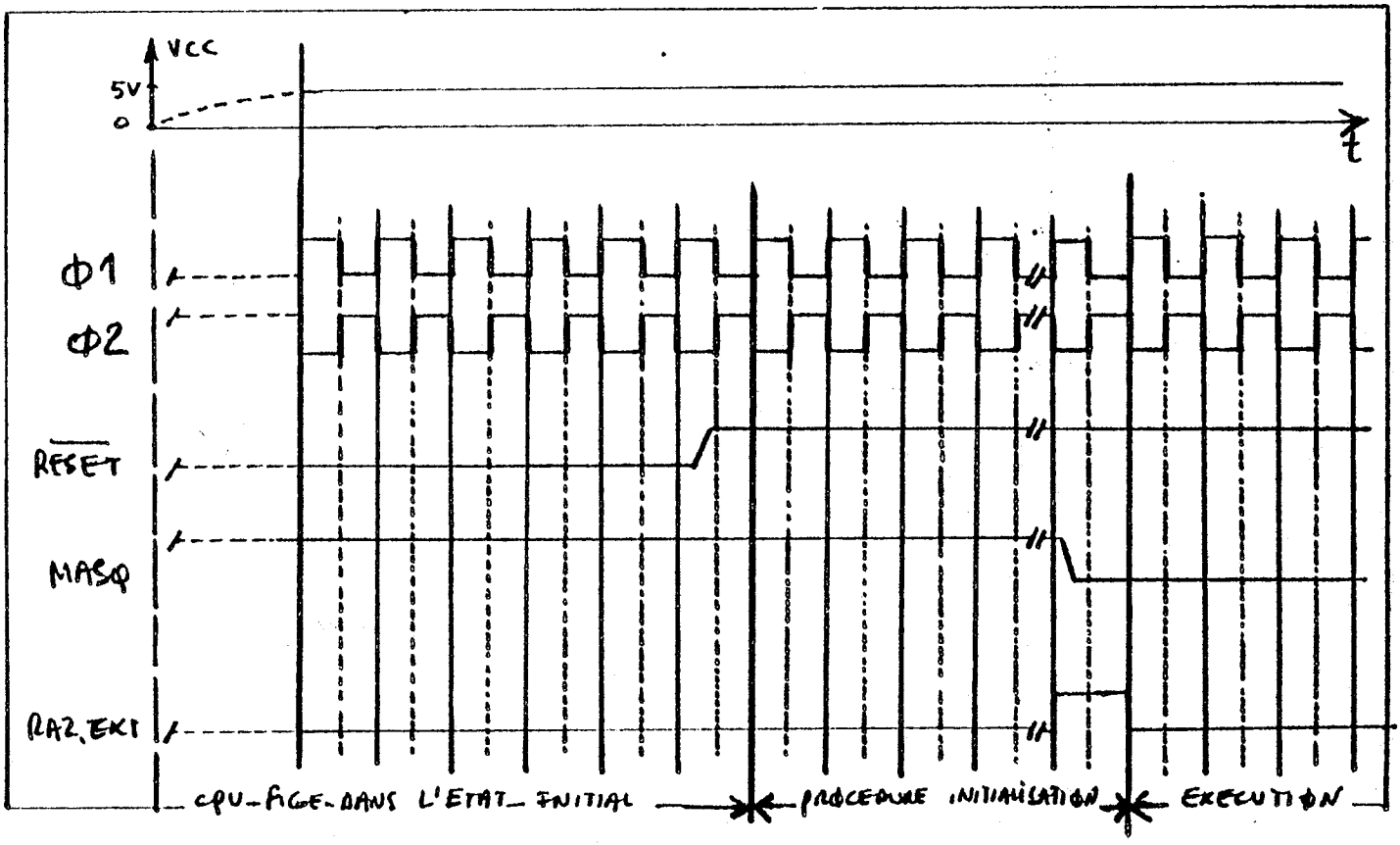


FIG.81:

Diagramme des temps des masques externes (NMI, IRQ).



4. NOTATIONS ET SYMBOLES UTILISES DANS LE PROJET 6800s :

Symboles pour les variables (registres internes, mémoires,...)

A = ACCA /8bits/ : Accumulateur A
B = ACCB /8bits/ : Accumulateur B
ACCX /8bits/ : Accumulateur A ou B.

X / 16 Bits/ : Registre d'index
XH / 8 bits/ : Octet de poids fort du registre d'index
XL / 8 bits/ : Octet de poids faible du registre d'index.

PC / 16 bits/ : Compteur ordinal
PCH/ 8 bits/ : Octet de poids fort du compteur ordinal
PCL/ 8 bits/ : Octet de poids faible du compteur ordinal

SP / 16 bits/ : Pointeur de pile
SPH/ 8 bits/ : Octet de poids fort du pointeur de pile
SPL/ 8 bits/ : Octet de poids faible du pointeur de pile

M / 8 bits/ : Position mémoire situé à l'adresse M.
TAMP/ 8 bits/ : Tampon de l'octet poids fort de l'adresse
RI / 8 bits/ : Registre instruction

RCC/ 8 bits/ : Registre des codes condition
H... Bit 5 du RCC : demi retenue
I... BIt 4 du RCC : masque d'interruption (IRQ)
N... Bit 3 du RCC : indicateur du signe
Z... Bit 2 du RCC : indicateur du résultat nul
V... Bit 1 du RCC : débordement en complément à 2
C... Bit 0 du RCC : retenue sortante.

UAL : Unité arithmétique et logique
DAA : Circuit de correction du calcul décimal
DECAL-D: Circuit de décalage droit
$\overline{DB}$: Circuit pour complémenter le bus DB bit à bit.

Lorsqu'après le nom d'un registre ou d'un accumulateur on ajoute l'indice n, cela signifie qu'il s'agit du $n^{ème}$ bit du registre ou de l'accumulateur considéré.

INC /16 bits/ : incrémenteur (+1), (+0) ou (-1)
INCH / 8 bits/ : incrémenteur poids fort
INCL / 8 bits/ : incrémenteur poids faible

T.INCH/8 bits, dynamiques/ : tampon de INCH
T.INCL/ " " : tampon de INCL
T.IN / " " : tampon d'entrée du bus-D
T.OUT / " " : tampon de sortie du bus-D
T.UAL / " " : tampon de l'UAL

Symboles pour l'UAL

$\wedge$: ET LOGIQUE
v : OU LOGIQUE
+ : addition arithmétique
$\oplus$: OU exclusif logique

SYMBOLES POUR LES TABLEAUX DES CODES OPERATIONS INVALIDES

OP	:	Code opération (en Hexadécimal)
∿	:	nombre de cycles d'horloge
#	:	nombre d'octets par instruction
S	:	Bit toujours mis à 1 (set)
R	:	Bit toujours mis à 0 (Reset)
(-)	:	Bit non modifié
↑	:	bit mis à 1 si test vrai, mis à 0 sinon
↓	:	bit mis à 0 si la retenue générée par l'UAL est C7 = 1, mis à 1 si la retenue générée par l'UAL est C7 = 0 (Tous les calculs dans l'UAL sont faits en complément à 2).

Les bits suivants sont mis à 1 si le test est vrai, sinon ils sont mis à 0 (valeurs des constantes en Hexadécimal),

(1)	Bit N....Test	:	01 <= ACCX <= 80 ?
(2)	Bit N....Test	:	ACCX = 80 ?
(3)	Bit C....Test	:	ACCX ≠ 00 ?
(4)	Bit Z....Test	:	ACCX = 00 ou 01 ?
(5)	Bit V... Test	:	ACCX (0) = 1 ?
(6)	Bit C....Test	:	ACCX (0) = 1 ?
(7)	Bit V....Test	:	Prend la valeur de N ⊕ C après décalage
(8)	Bit N....Test	:	Bit de signe de l'octet poids fort = 1 ?
(9)	Bit N....Test	:	Résultat négatif (Bit 1 C = 1) ?
(10)	Bit V....Test	:	Dépassement en complément à 2 dans la soustraction des octets de poids fort ?

ANNEXE .3 - EVALUATIONS CONCRETES SUR LE 6800S

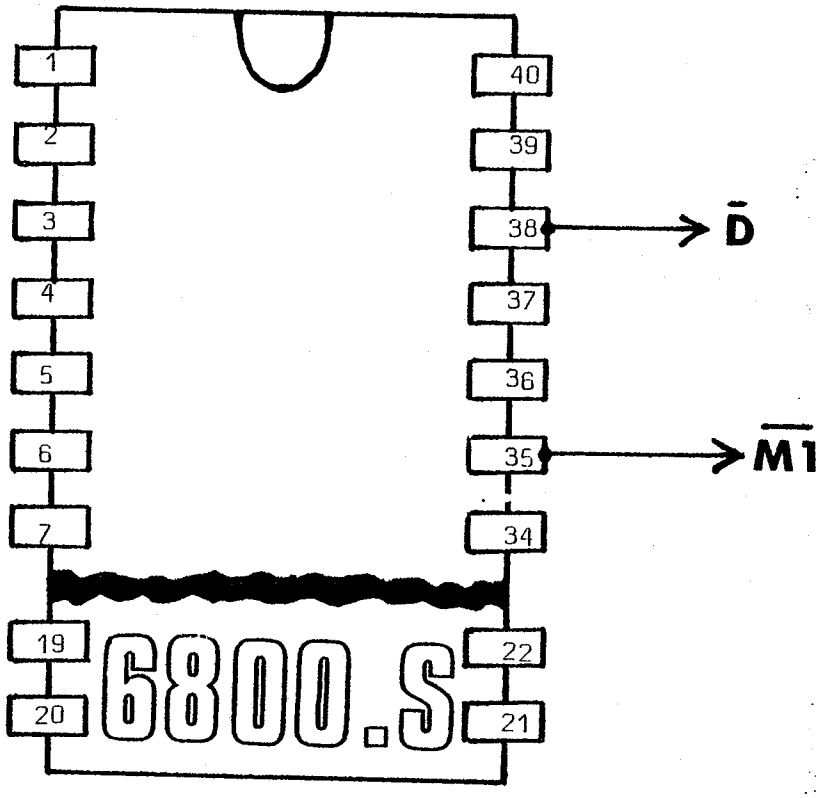
La réalisation technique du 6800S se fait à partir des masques du circuit intégré de la version actuelle du 6800.

Les modifications des masques permettent l'insertion d'un mécanisme de détection des pannes par codes invalides et la suppression des anomalies de fonctionnement du 6800.

Nous présentons dans ce qui suit, les différentes évaluations (logique, électrique, topologique) concernant la correction des défauts du 6800 ainsi que les caractéristiques du bloc [ANOMALIE] ce qui permet la réalisation des masques du 6800S.

1. - Brochage du 6800S

Toutes les broches du 6800S sont identiques à celles du 6800
à l'exception des broches 35 et 38 qui ne sont pas utilisées sur le 6800.



Signaux supplementaires du 6800.S /6800 -FIG.82

La broche 35 ($\overline{M1}$) est une sortie qui indique le premier cycle de l'instruction en cours.
La broche 38 ($\overline{D}$) est une sortie qui indique la détection d'un code invalide.
Ces deux signaux sont actifs au niveau bas.

L'utilité du signal [$\overline{D}$] est indéniable. Il en est de même du signal [$\overline{M1}$] qui, non seulement facilite l'utilisation du 6800S, mais lui donne plus de possibilités.

En effet,

- il permet la synchronisation des signaux externes (par exemple: les signaux d'interruptions),
- il permet l'installation d'un mecanisme de protection memoire (differentiation entre lecture données, lecture instructions),
- il permet l'extention de codes (decodage externe).

Notons que si le 6800.S rentre dans un sequence de sauvegarde pendant le traitement d'une interruption materielle ou pendant une sequence d'arret par HALT, le signal ($\overline{M1}$) n'est pas active.La sequence de Programme de la figure suivante (contenant un code invalide) permet de visualiser l'activation de ces deux signaux de controle.

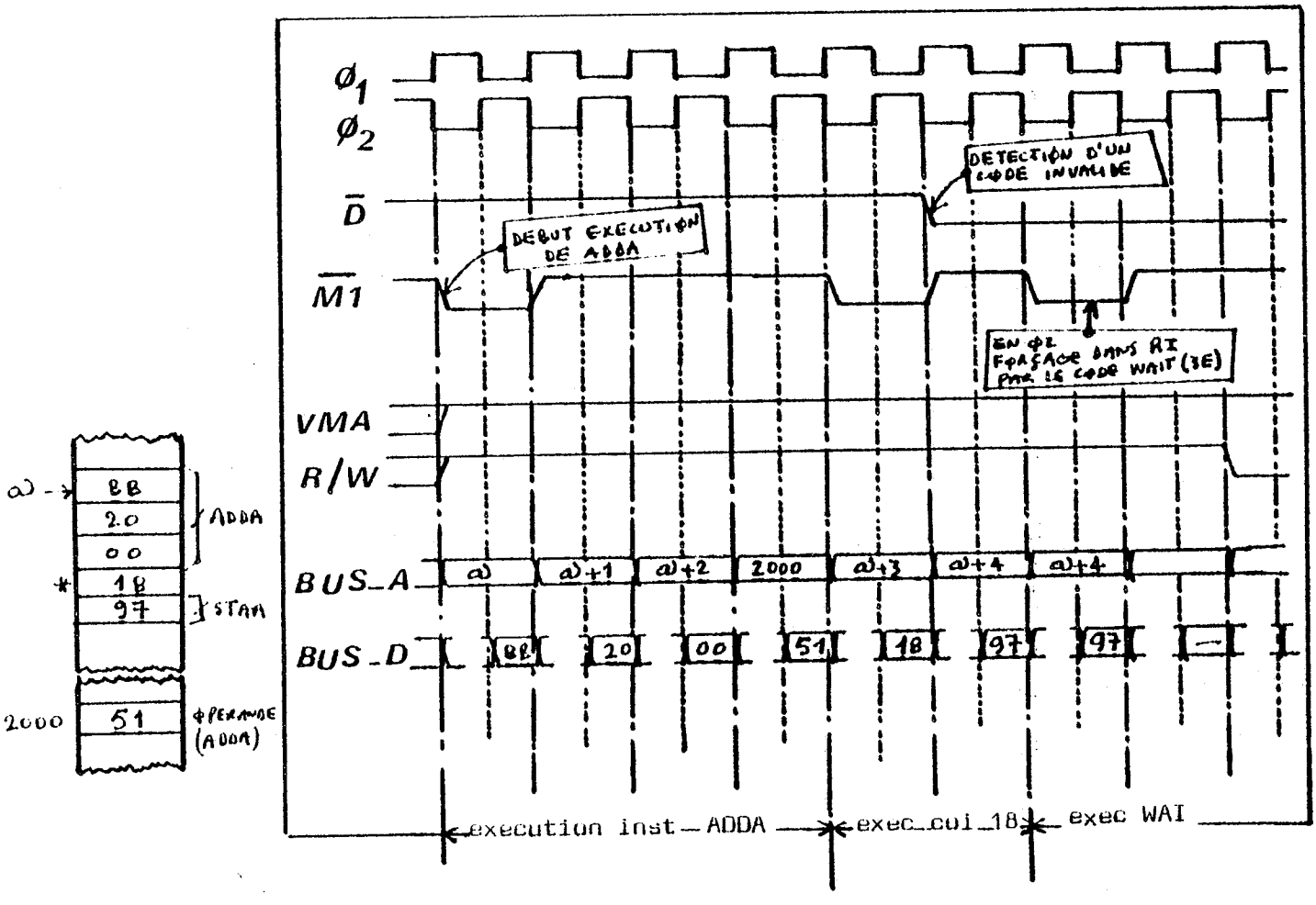


FIG.83: Diagramme des temps des signaux de controle ($\overline{D}$), ($\overline{M1}$) du 6800.S

---

* Le code invalide 18 est dû à une erreur ou à une panne materielle.

2. - Evaluation logique : bloc [ANOMALIE]

Le mécanisme de détection de pannes par codes invalides (*) est bâti autour d'un PLA qui permet la reconnaissance de l'un quelconque des 59 codes invalides. Les monomes qui constituent la première couche logique du PLA sont au nombre de 15.

	PROFILS DU PLA								CODES INVALIDES DETECTES							
Monomes	I 7	I 6	I 5	I 4	I 3	I 2	I 1	I 0								
M1	0	0	0	0	0	0	0	0	00							
M2	0	0	0	-	0	0	1	-	02	03	12	13				
M3	0	0	0	-	0	1	0	-	04	05	14	15				
M4	0	0	0	1	1	1	-	-	1C	1D	1E	1F				
M5	0	0	-	1	1	0	-	0	18	1A	38	3A				
M6	0	0	1	0	0	0	0	1	21							
M7	0	1	-	-	0	0	-	1	41	42	51	52	61	62	71	72
M8	0	0	1	1	1	1	0	-	3C	3D						
M9	0	1	-	-	0	0	1	0	45	55	65	75				
M10	0	1	-	-	1	1	0	0	4B	5B	6B	7B				
M11	0	1	0	-	1	1	1	0	4E	5E						
M12	1	-	-	-	0	0	1	1	83	93	A3	B3	C3	D3	E3	F3
M13	1	-	0	0	-	1	1	1	87	8F	C7	CF				
M14	1	0	0	1	1	1	0	1	9D							
M15	1	1	-	-	1	1	0	-	CC	CD	DC	DD	EC	ED	FC	FD

Monomes permettant la détection des 59 codes invalides du 6800.

(*) Bloc ANOMALIE .

Le câblage suivant représente le bloc [ANOMALIE] qui réalise le contrôle des opérations qui suivent la détection d'un code invalide.

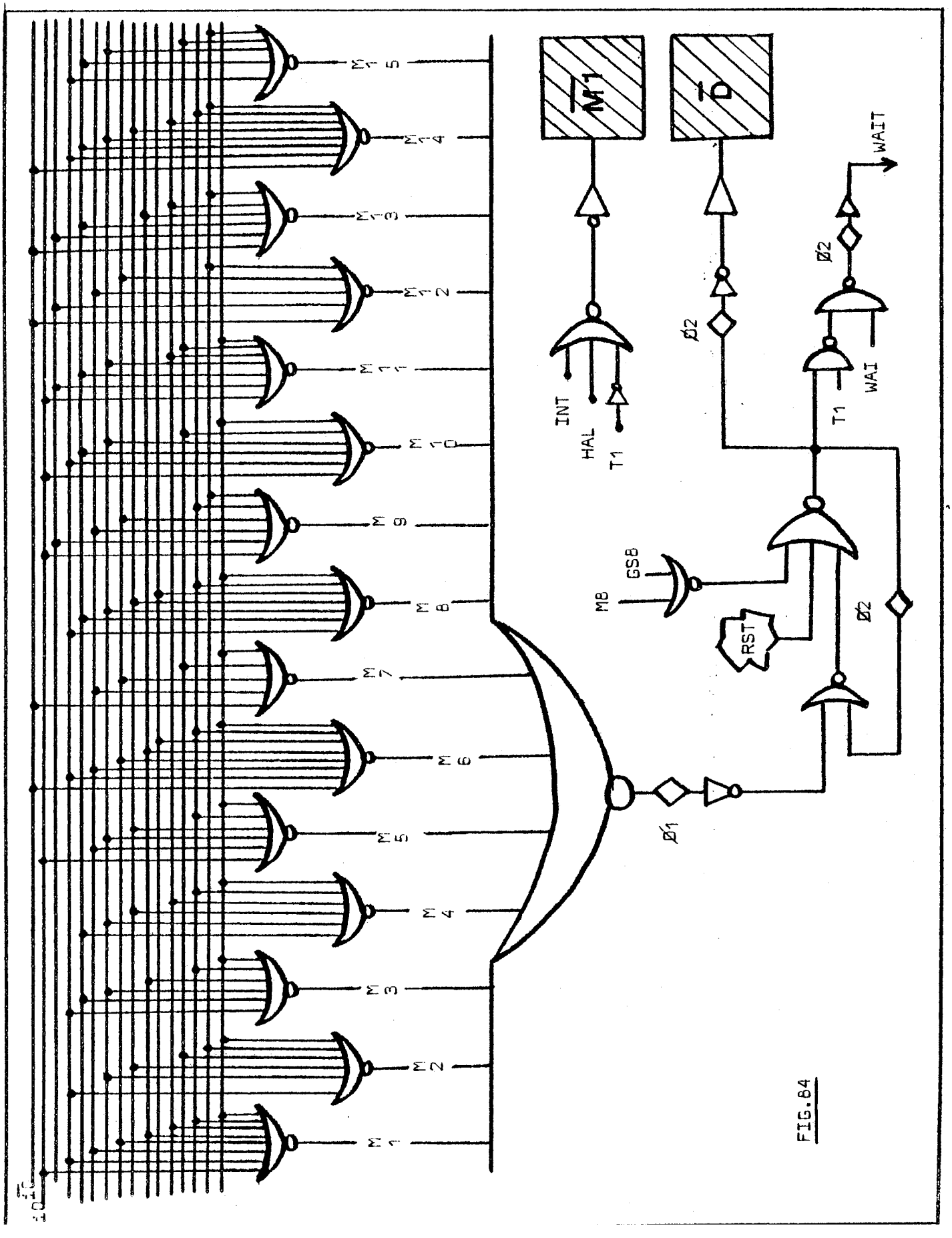


FIG.84

3. - Evaluation électrique

La simulation électrique a été faite pour les principaux blocs de la partie contrôle et en particulier pour les parties consernant la correction des anomalies du 6800.

Les différentes charges (capacités, résistances) des lignes de commandes de la partie contrôle ont été calculées à partir de l'observation du circuit intégré du 68B00.

Les différentes simulations ont été faites dans des conditions de température et d'alimentation diverses, notamment pour les cas extrêmes de fonctionnement :

- température ambiante, T = 343°K,
- tension, VCC = 4,75 volts.

Le simulateur qui a été utilisé pour cette étude est le MSINC (*).

Nous présentons dans ce qui suit, le bloc [ANOMALIE] et la bascule de mémorisation de l'interruption [NMI]. Nous présentons également la simulation d'un point de la bascule du [NMI].

Des résultats plus complets et plus détaillés sont présentés dans (29).

(*) De l'Université de Stanford.

4. - Evaluation électrique sur le bloc [ANOMALIE]

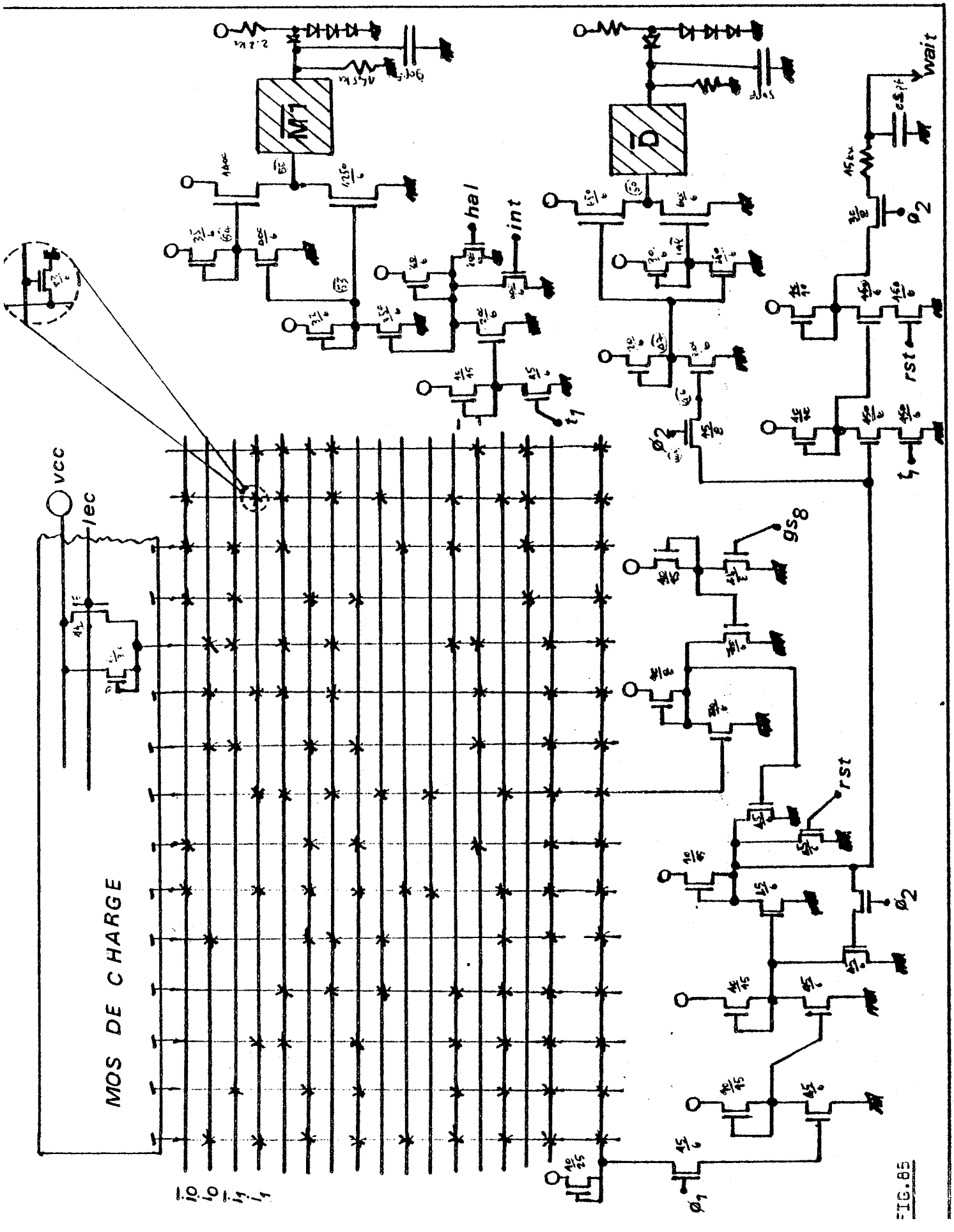


FIG.85

3.2_ Evaluation electrique sur la bascule NMI

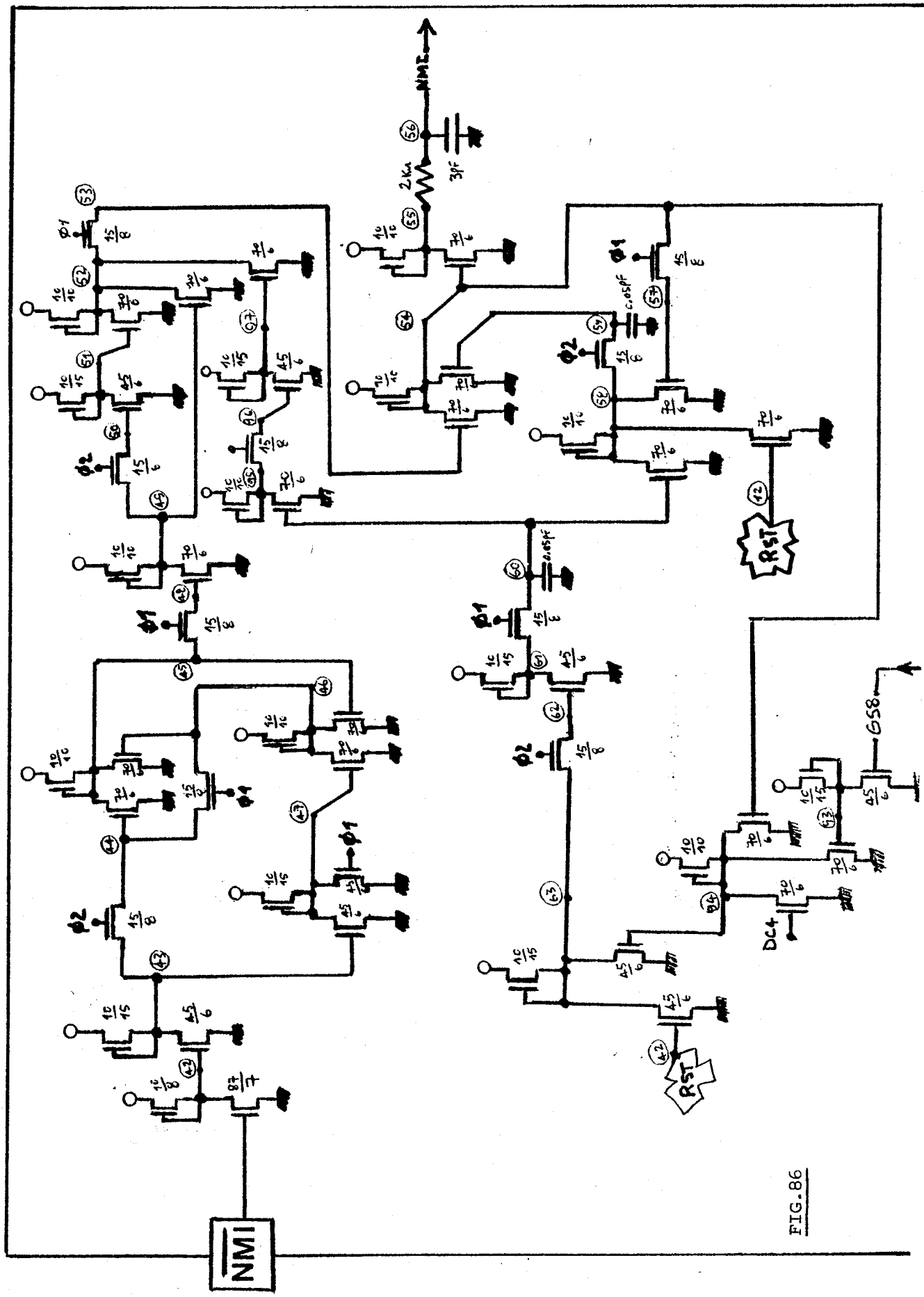


FIG.86

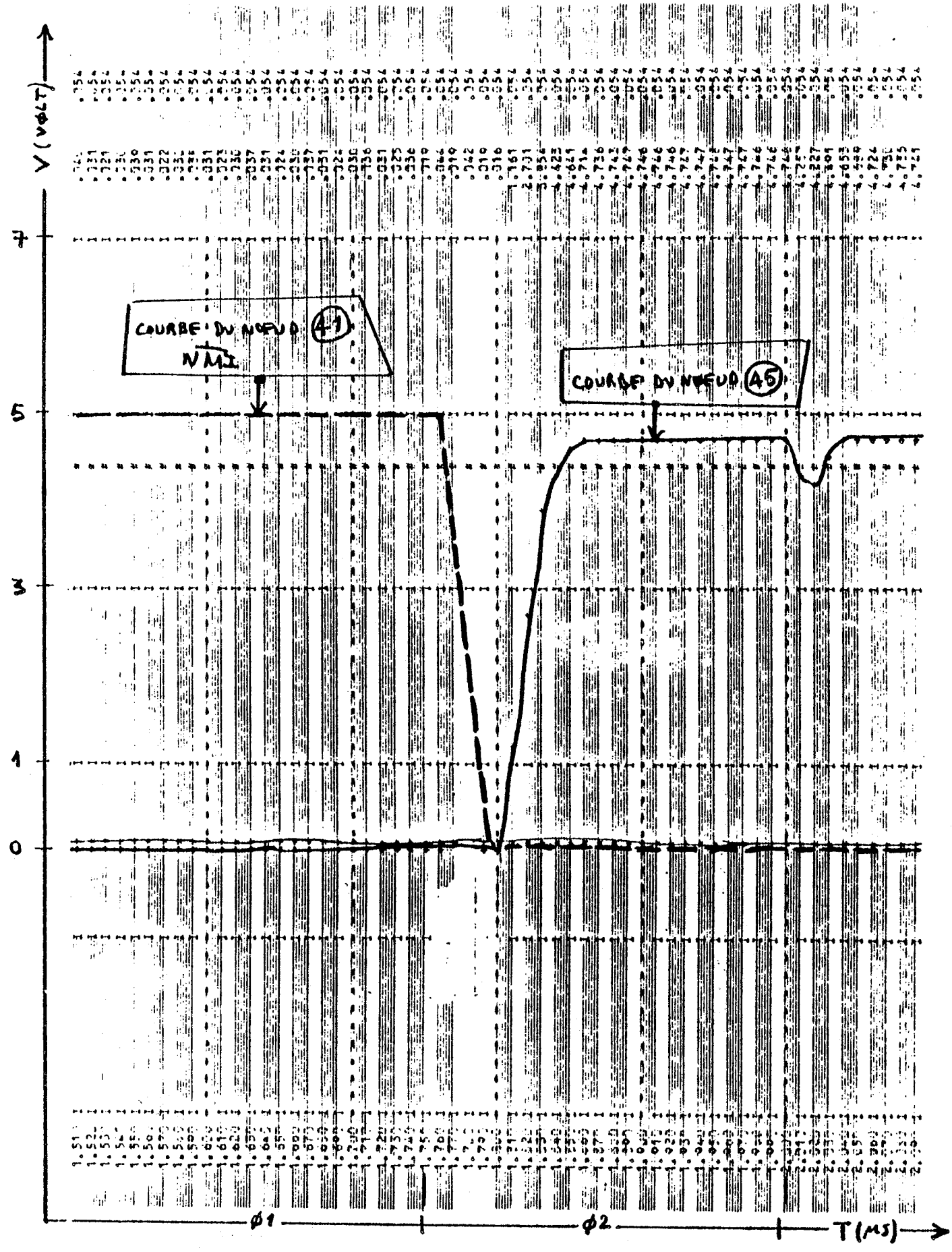


FIG.87- Simulation de la barriere d'echantillonnage de la bascule du NMI /6800.S

Les conditions de la simulation sont; T = 320°K

VCC= 5 Volts.

3. - Evolution topologique

3.1. - Correction de l'anomalie.2

L'anomalie.2 est la plus simple à corriger.
En effet, il suffit de masquer les conditions (DC5, DC6, DC7) issues du décodeur par le signal $T2$ qui indique le deuxième cycle de la séquence Acquisition.

Ceci se traduit sur les masques du circuit intégré par l'adjonction de trois transistors MOS signaux commandés par la grille ($\overline{T_2}$).
La géométrie de ces trois MOS signaux est : $\gamma s = \frac{13}{6}$

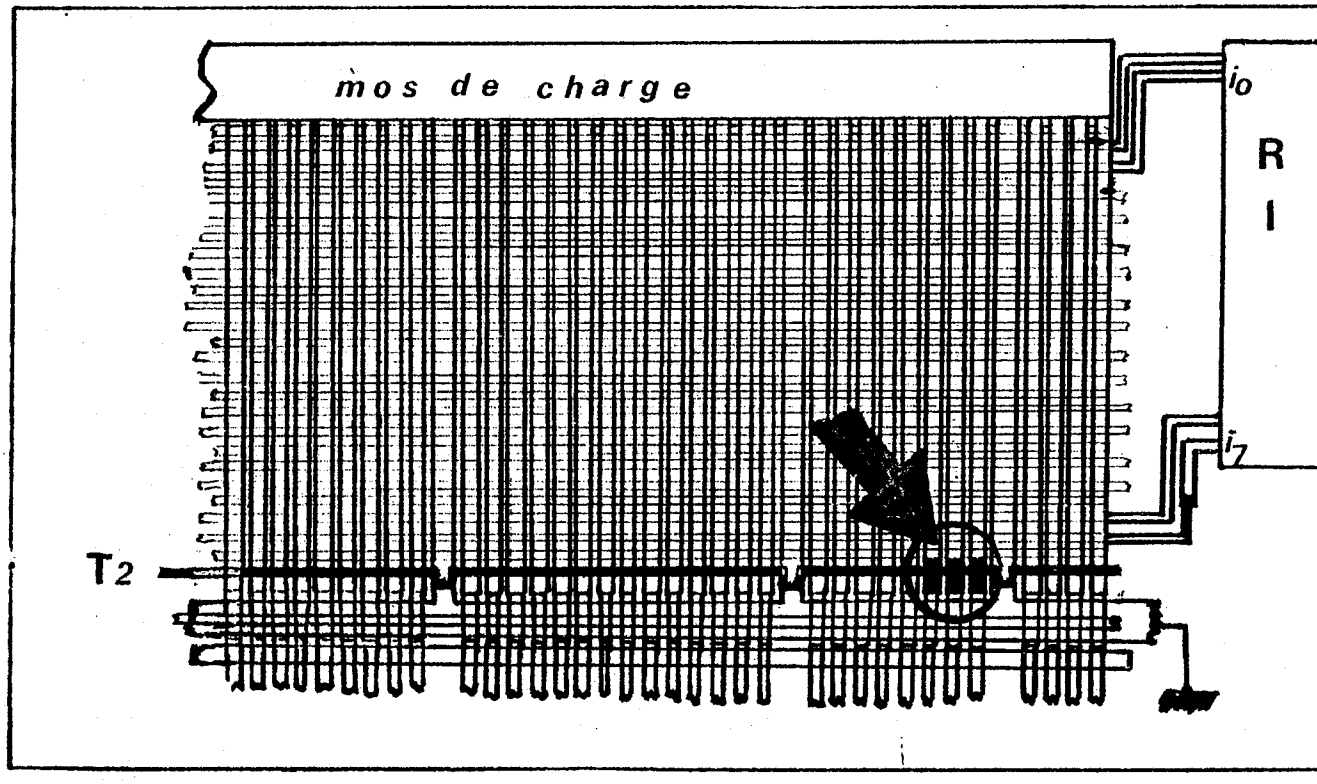


FIG. 88 - Suppression de l'anomalie 2 au niveau des masques.

Note : La géométrie des MOS de charges correspondant aux monomes du décodeur DC5, DC6, DC7 à modifier.

3.2. - Evaluation globale sur la topologie du 6800S

Les modifications au niveau des masques du circuit intégré du 6800 (version actuelle) ne sont pas très importantes.

L'augmentation de la surface des masques du 6800S reste assez faible par rapport à celle du 6800 : 5,8%.

Jusqu'à présent les masques du 6800S n'ont pas été réalisés. Seules des estimations concernant la topologie de la zone à modifier ont été faites. Les deux figures ci-après exposent les différentes estimations qui ont été réalisées.

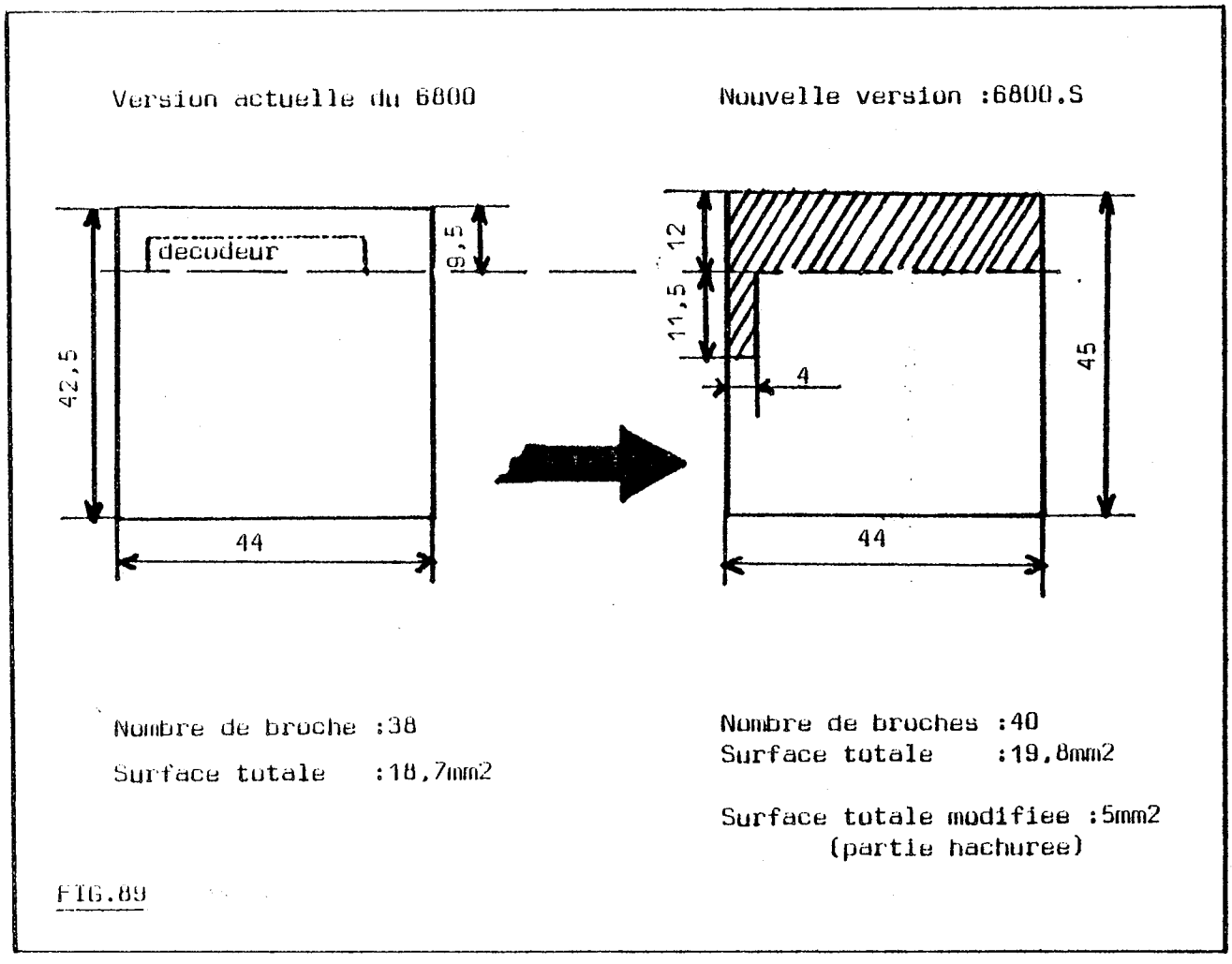
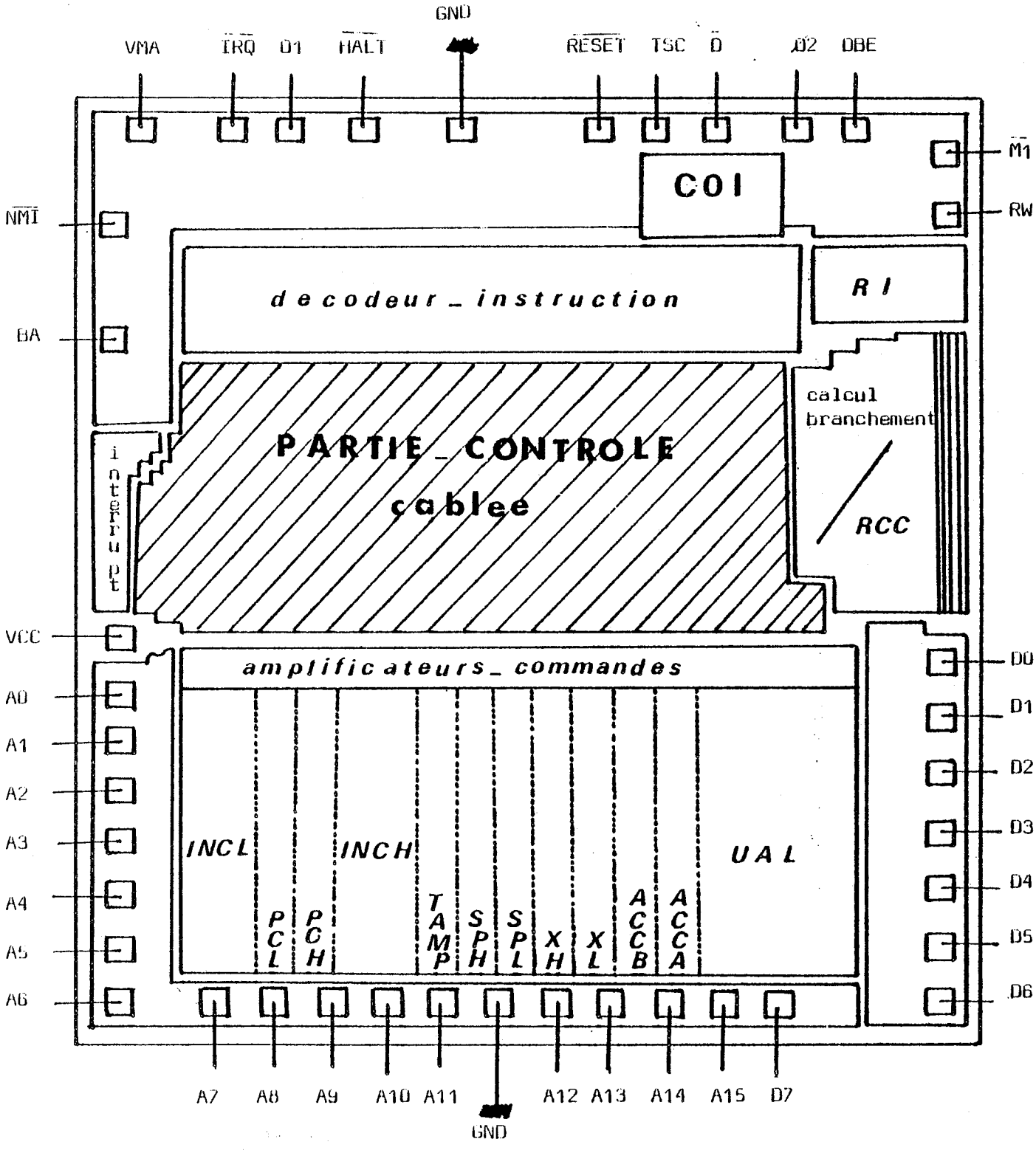


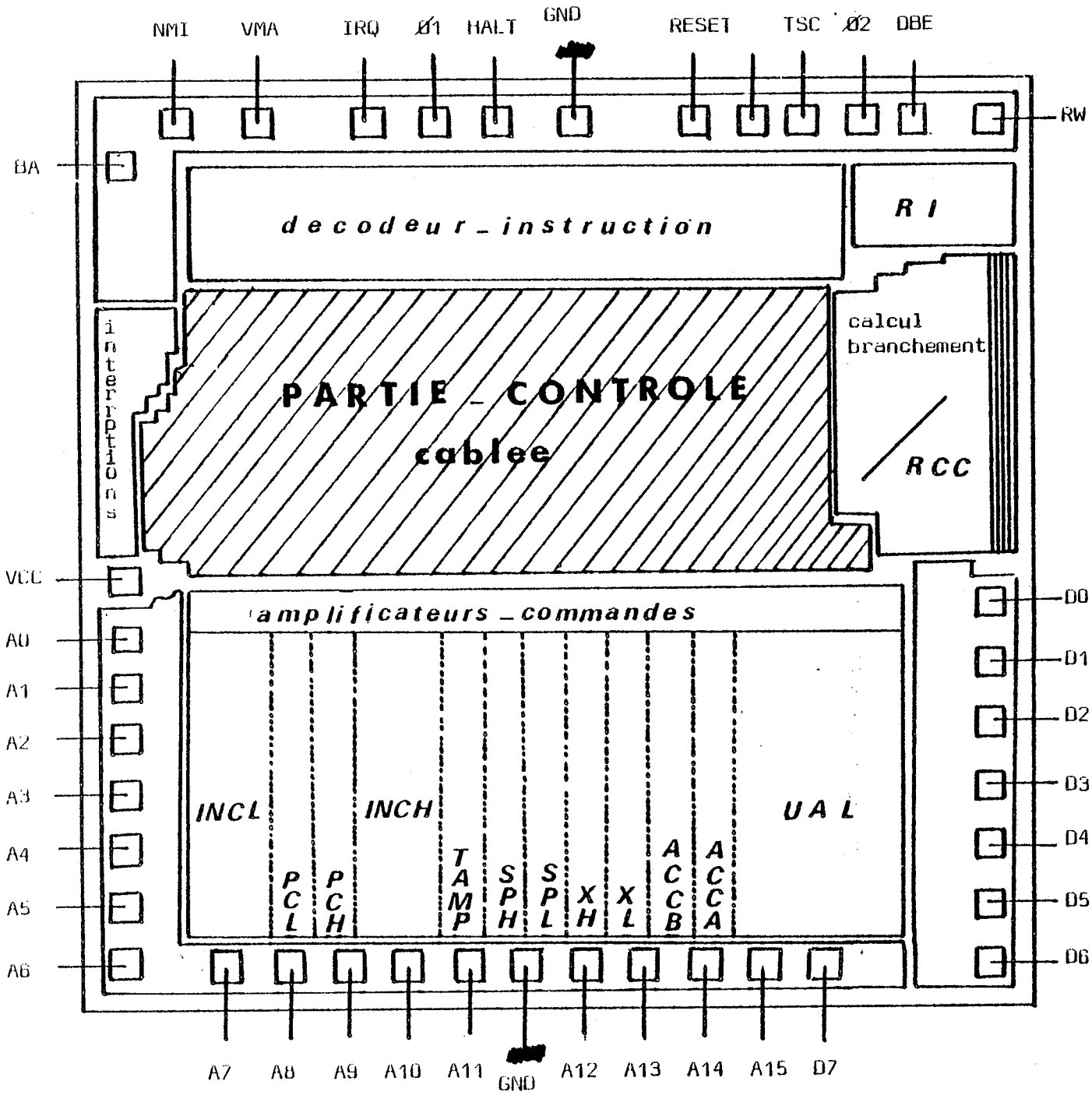
FIG.89

6800 S



nb pluts =40
surface =19,8mm2
surface modifiee =5mm2

6800



nb pluts =38
surface =18,7mm2

BIBLIOGRAPHIE

(1) F.ANCEAU
CONTRIBUTION A L'ETUDE DES SYSTEMES HIRARCHISES DE RESSOURCES DANS L'ARCHITECTURE DES MACHINES INFORMATIQUES.
/these d'etat, inpg, grenoble 5.dec.74

(2) F.ANCEAU
MECANISMES PRIMITIFS DE SYNCHRONISATION AU NIVEAU MATERIEL.
/2nd colloque international sur les systemes d'exploitation iria, 2/4.oct.79

(3) F.ANCEAU
L'EVOLUTION DES MICROPROCESSEURS ET LEUR INFLUENCE SUR L'EVOLUTION DES SYSTEMES.
/real time data, oct.79

(4) R.BIANCHI
IMPLANTATION DE LA LOGIQUE ALEATOIRE D'UN CIRCUIT INTEGRE.
/rapport de dea.microelectronique, inpg, juin.81

(5) C.ANDRE
METHODE DE CONCEPTION ASSISTE PAR ORDINATEUR DES SYSTEMES LOGIQUES A EVOLUTION SIMULTANEES.
/these de docteur.ingenieur, universite de nice

(6) D.BORRIONE
LANGAGES DE DESCRIPTION DE SYSTEMES LOGIQUES. PROPOSITION POUR POUR UNE METHODE FORMELLE DE DEFINITION.
/these d'etat, inpg, 1.jui.81

(7) M.R.BARBACCI
A COMPARISON OF REGISTER TRANSFER LANGUAGES FOR DESCRIBING COMPUTERS AND DIGITAL SYSTEMS.
/ieee trans.comput,vol.c.24, n°2, feb.75

(8) H.D.CAPLENER
TOP DOWN APPROACH TO LSI SYSTEM DESIGN.
/computer design, aug.74

(9) Y.CHU
INTRODUCING CDL.
/ieee trans.comput,vol.7, n°12,dec.74

(10) C.R.CLARE
DESIGNING LOGIC SYSTEMS USING STATE MACHINES.
/m.c graw hill, new-york.73

(11) J.M.COSTA ALVES MARQUES
MOSAIC: UNE METHODOLOGIE DE CONCEPTION POUR LES CIRCUITS SYSTEMES VLSI
/these docteur.ingenieur, inpg, sep.80

(12) B.COURTOIS
TEST FONCTIONNEL DE LA PARTIE CONTROLE D'UNITES CENTRALES INTEGREES
/rr.203, inpg, mai.80

(13) B.COURTOIS
CODAGE DES INSTRUCTIONS D'UNE UNITE CENTRALE INTEGREE EN VUE DE LA DETECTION DE PANNES.
/rr.207, inpg, juil.80

(14) E.DACLIN, M.BLANCHARD
SYNTHESE DES SYSTEMES LOGIQUES.
/cepadus.edit, dec.76

(15) D.DEDEURWAERDER, P.MOUGEAT
PRESENTATION DU FONCTIONNEMENT INTERNE DU MICROPROCESSEUR 68000.
/note interne efcis, jan.81

(16) M.FILLON
METHODE D'IMPLANTATION DE LA LOGIQUE ALEATOIRE.
/rapport de dea.microelectronique, inpg, juin.81

(17) D.GUILIANI
WILL DESIGN TOOL CATCH UP TO VLSI DESIGNERS NEEDS.
/16th design autom.conf,san-diego.california, june.79

(18) V.M.GLUSKOV, A.A.LETICHEVSKI
THEORY OF ALGORITHMS AND DISCRETE PROCESSORS.
/advances information, systems science, vol.1

(19) D.HIGTOWER
CAN CAD MEET THE VLSI DESIGN PROBLEMS OF THE 80'S.
/16th design autom.conf,san-diego.california,june.79

(20) L.H.JONES
THE ROLE OF INSTRUCTION SEQUENCING IN STRUCTURED MICROPROGRAMMING.
/ieee trans.comput,aug.74

(21) M.W.LARKIN
IMPACT OF TECHNOLOGY ON THE DEVELOPPMENT OF VLSI.
/vlsi.81, accademic press

(22) B.LATTIN *(intel corporation)*
VLSI DESIGN, METHODOLOGY, THE PROBLEMS OF THE 80'S FOR MICROPROCESSOR DESIGN.
/16th design auto.conf,san-diego.california,june.79

(23) B.LECUSSAN
EMULATION GENERALISEE; DETERMINATION D'OUTILS MATERIELS DANS L'INTERPRETATION DE LANGAGES EVOLUES.
/these docteur 3ē.cycle, universite.toulouse, mars.77

(24) F.P.MAISON
ETUDE DES PROBLEMES LIES A LA CONCEPTION DES SYSTEMES INFORMATIQUES EN LSI.
/these d'etat, inpg, 25.sept.78

(25) C.A.MEAD, L.A.CONWAY
INTRODUCTION TO VLSI SYSTEMS.
/addison wesley, 78

(26) J.MERMET
ETUDE METHODOLOGIQUE DE LA CONCEPTION ASSISTEE PAR ORDINATEUR DES SYSTEMES LOGIQUES: CASSANDRE.
/these d'etat, inpg, 10. avr.73

(27) K.D.MUELLER GLASER, L.LERACH
A GENERAL CELL APPROACH FOR SPECIAL PURPOSE VLSI CHIPS.
/vlsi.81, academic press

(28) M.NEMMOUR
CONCEPTION ET REALISATION AUTOMATIQUE D'UN COMPOSANT FONCTIONNEL: SYSTEME DE DECODAGE DU MICROPROCESSEUR P.68.
/rapport de dea.genie-infor, inpg, juin.78

(29) M.NEMMOUR
ETUDE DU FONCTIONNEMENT INTERNE DES MICROPROCESSEURS 6800.
/contrat edf-ensimag n°511.78.1.0, mai.79

(30) M.NEMMOUR
STRUCTURES "BLOCK-SLICE": UNE METHODOLOGIE DE CONCEPTION DES CIRCUITS INTEGRES VLSI.
/rapport interne efcis, juin.81

(31) N.KAWATO, T.SAÏTO, F.MARUJAMA, T.UEHARA
DESIGN AND VERIFICATION OF LARGE SCALE COMPUTERS BY USING DDL.
/16th design auto.conf,san-diego.california,june.79

(32) G.NOGUEZ
ETUDE D'UN MODEL TEMPOREL DES SYSTEMES SEQUENTIELS.
/these d'etat, inst.programmation paris.VI, sept.75

(33) M.OBREBSKA
ETUDE COMPARATIVE DES DIFFERENTES METHODOLOGIES DE CONCEPTION DES PARTIES CONTROLE DES MICROPROCESSEURS.
/rr.217, inpg, oct.80

(34) S.S.PATIL, J.B.DENNIS
THE DESCRIPTION AND REALIZATION OF DIGITAL SYSTEM COMPUTATION STRUCTURES.
/sixth annual ieee,comp.society, oct.72

(35) M.SILVA-SUAREZ
CONTRIBUTION A LA SYNTHESE PROGRAMMEE DES AUTOMATISMES LOGIQUES.
/these de docteur-ingenieur, inpg, juin.78

(36) R.REIS
ETUDE DE L'ARCHITECTURE INTERNE DU Z8000 (PARTIE CONTROLE).
/rapport de dea.genie-info, inpg, sept.80

(37) C.L.SEITZ
GRAP REPRESENTATION OF LOGICAL MACHINES.
/caltech

(38) C.E.SHANON
SYMBOLIC ANALYSIS OF RELAY AND SWITCHING CIRCUITS.
/caltech

(39) J.P.SCHOELLKOPF
MICROPROGRAMMING: A STEP OF A TOP DOWN DESIGN METHODOLOGY.
/7th annualwork shop on micro.,paco.alto, oct.74

(40) J.P.SCHOELLKOPF
MACHINE PASC-HLL: DEFINITION D'UNE ARCHITECTURE PIPE-LINE POUR UNE UNITE CENTRALE ADAPTEE AU LANGAGE PASCAL.
/these docteur 3è.cycle, inpg, juin.77

(41) A.B.TUCKER, M.S.FLYNN
DYNAMIC MICROPROGRAMMING: PROCESSEUR ORGANISATION AND MICROPROGRAMMING.
/conf of acm, vol.14 n°4, april.74

(42) W.M.VANCLEEMPUT
A HIERARCHICAL LANGUAGE FOR THE STRUCTURAL DESCRIPTION OF DIGITAL SYSTEMS.
/14th design auto.conf, n.orleans, june.77

(43) W.M.VANCLEEMPUT
COMPUTER HARDWARE DESCRIPTION LANGUAGE AND THEIR APPLICATIONS.
/16th design auto.conf, san-diego.california, june.79

(44) R.WAXMAN
VLSI A DESIGN CHALLENGE.
/16th design auto.conf, san-diego, california, june.79

(45) W.WEMANN *(motorola inc.)*
CAD SYSTEM FOR VLSI.
/16th design auto.conf, san-diego.california, june.79

(46) J.D.WILLIAMS
STICKS A NEW APPROACH TO LSI DESIGN.
/these de msc massachusset.inst of technology, june.77

(47) N.WIRTH
ALGORITHMS + DATA STRUCTURES = PROGRAMMS.
/prentice hall.inc, englewood cliffs, 76

STITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année Universitaire
1980-1981

ésident : M. Philippe TRAYNARD

ce-Présidents : M. Georges LESPINARD
M. René PAUTHENET

PROFESSEURS DES UNIVERSITES

CEAU François	Informatique fondamentale et appliquée
SSON Jean	Chimie Minérale
IMAN Samuël	Electronique
OCH Daniel	Physique du Solide - Cristallographie
IS Philippe	Mécanique
NNETAIN Lucien	Génie Chimique
NNIER Etienne	Métallurgie
UVARD Maurice	Génie Mécanique
ISSONNEAU Pierre	Physique des Matériaux
YLE-BODIN Maurice	Electronique
ARTIER Germain	Electronique
ENEVIER Pierre	Electronique
ERADAME Hervé	Chimie Physique Macromoléculaire
ERUY Arlette	Automatique
IAVERINA Jean	Biologie, biochimie, agronomie
HEN Joseph	Electronique
UMES André	Electronique
RAND Francis	Métallurgie
RAND Jean-Louis	Physique Nucléaire et Corpusculaire
LICI Noël	Electrotechnique
ULARD Claude	Automatique
YOT Pierre	Métallurgie Physique
ANES Marcel	Electrotechnique
UBERT Jean-Claude	Physique du Solide - Cristallographie
URDAIN Geneviève	Traitement du Signal
COUME Jean-Louis	Géophysique - Traitement du Signal
NCIA Roland	Electronique - Automatique
SIEUR Marcel	Mécanique
SPINARD Georges	Mécanique
NGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
REAU René	Mécanique
RET Roger	Physique Nucléaire Corpusculaire
RIAUD Jean-Charles	Chimie-Physique
UTHENET René	Physique du Solide - Cristallographie
RRET René	Automatique
RRET Robert	Electrotechnique
AU Jean-Michel	Mécanique
LOUJADOFF Michel	Electrotechnique
UPOT Christian	Electronique - Automatique
MEAU Jean-Jacques	Electrochimie - Corrosion
BERT André	Chimie appliquée et des matériaux
BERT François	Analyse numérique
BONNADIERE Jean-Claude	Electrotechnique
UCIER Gabrielle	Informatique fondamentale et appliquée

.../...

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

PROFESSEURS DES UNIVERSITES

MmeSCHLENKER Claire	Physique du Solide - Cristallographie
MM SCHLENKER Michel	Physique du Solide
SOHM Jean-Claude	Chimie Physique
TRAYNARD Philippe	Chimie - Physique
VEILLON Gérard	Informatique fondamentale et appliquée
ZADWORNY François	Electronique

CHERCHEURS DU C.N.R.S. (Directeur et Maîtres de Recherche)

M FRUCHART Robert	Directeur de Recherche
MM ANSARA Ibrahim	Maître de Recherche
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DRIOLE Jean	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Doré	Maître de Recherche
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (Décision du Conseil Scientifique)

E.N.S.E.E.G.

MM ALLIBERT Michel	
BERNARD Claude	
CAILLET Marcel	
MmeCHATILLON Catherine	
MM COULON Michel	
HAMMOU Abdelkader	
JOUD Jean-Charles	
RAVAINE Denis	
SAINFORT	C.E.N.G.
SARRAZIN Pierre	
SOUQUET Jean-Louis	
TOUZAIN Philippe	
URBAIN Georges	Laboratoire des Ultra-Réfractaires ODEILLO

E.N.S.M.S.E.

MM BISCONDI Michel	
BOOS Jean-Yves	
GUILHOT Bernard	
KOBILANSKI André	
LALAUZE René	
LANCELOT Francis	
LE COZE Jean	
LESBATS Pierre	
SOUSTELLE Michel	
THEVENOT François	

.../...

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

THOMAS Gérard
TRAN MINH Canh
DRIVER Julian
RIEU Jean

E.N.S.E.R.G.

BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

E.N.S.I.E.G.

BORNARD Guy
DESCHIZEAUX Pierre
BLANGEAUD François
HAUSSAUD Pierre
HOURDAIN Geneviève
LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G.

DELHAYE Jean-Marc

E.N.S.I.M.A.G.

COURTIN Jacques
LATOMBE Jean-Claude
LUCAS Michel
VERDILLON André

*
* *
*

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974

VU le rapport de présentation de

Monsieur François ANCEAU, Professeur

Monsieur Mohamed NEMMOUR

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTORAT DE TROISIEME CYCLE, Spécialité "Génie Informatique".

Fait à Grenoble, le 24 novembre 1981

Le Président de l'I.N.P.-G.

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble