Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche

researchportal.unamur.be

UNIVERSITÉ DE NAMUR
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Architecture de systèmes et Microprogrammation Dynamique. Projet EPRON (Experimental PROcessor Namur)

Demarteau, J.

Award date:
1973

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. May. 2026

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

1972-1973

Architecture de systèmes
et
Microprogrammation Dynamique

Projet EPRON

(Experimental PROcessor Namur)

J. DEMARTEAU

Mémoire présenté en vue de l'obtention
du grade de Licencié et Maître en Informatique

JURY : J. BRUNIN

De même que son ignorance grammaticale
ne saurait rendre malheureux le cheval,
la Folie ne fait point le malheur de l'homme
puisqu'elle est conforme à sa nature.

Eloge de la Folie XXXII

ERASME

Nous tenons à remercier Monsieur BRUNIN pour sa direction enthousiaste. Tout au long du travail il nous a confronté aux problèmes bien concrets et proposé des ébauches de solutions.

Nous devons à Monsieur JACQUES tout l'étude du hardware et la faisabilité de l'EPRON. Au fil des mois il a mis au point avec nous, la définition sémantique du hardware et vérifié différents micro-programmes. Nous lui témoignons ici notre estime et notre reconnaissance.

Nous remercions également Monsieur CHERTON pour les mécanismes qu'il nous a proposé et l'intérêt qu'il a manifesté dans leur mise au point.

Messieurs BIENVENU et CASSONNET de la C.H.B. nous ont introduit avec remarquablement de précision dans les domaines de l'architecture de système et de la Micro-programmation. Nous leur témoignons toute notre reconnaissance.

Nous relevons d'une façon toute particulière le travail de Madame WIBIN. Elle a donné son visage à ce travail, chacun au fil des pages pourra apprécier le soin qu'elle y a mis.

Les quelques incohérences de notation nous sont entièrement imputables et nous prions le lecteur de bien vouloir nous en excuser.

Namur, le 21 avril 1973.

SOMMAIRE

Chapitre 0 Introduction approfondie.

Chapitre 1 Définition du hardware.

Chapitre 2 Réalisation de micro-programmes.

Chapitre 3 Définition d'une architecture software-firmware dynamique.
Le macro assembleur supporté par firmware.

Conclusion.

Bibliographie.

Appendice. Schémas Hardware de l'EPRON.

Table des Matières.

Le travail présenté est le résultat d'expériences acquises au contact de concepteurs d'ordinateurs. La réflexion développée au cours de ces pages s'appuie sur l'étude de machines bien précises et plus spécialement sur l'EPRON, prototype développé à l'Institut d'informatique.

Le chapitre 0 présente le cadre et les lignes essentielles de la recherche, les Chapitres 1, 2 et 3 en développent quelques étapes fondamentales.

L'annexe présentant quelques schémas d'implantation visualise l'organisation hardware de l'EPRON.

CHAPITRE 0

INTRODUCTION APPROFONDIE.

1. Evolution historique.

2. Prospectives.

3. Les grands axes de la micro-programmation.

3.1. Contrôle de l'adresse.
3.2. Décode en champs.
3.3. Indépendance des mots de contrôle.
3.4. Lignes de recherches actuelles.
3.4.1 - Multiplication des niveaux.
3.4.2 - Dynamisme firmware.
3.4.3 - Aide à la réalisation de micro-programmes.

4. L'EPRON face à ses axes.

4.1. Contrôle de l'adresse.
4.2. Décode en champs.
4.3. Indépendance des mots de contrôle.

5. Macro assembleur de micro-programmes.

5.1. Principes.
5.2. Justification.
5.3. Réalisation.
5.4. Points précis d'implémentation.
5.5. Conclusion et Perspectives.

CHAPITRE 0

INTRODUCTION APPROFONDIE.

0. 1. EVOLUTION HISTORIQUE.

Deux touches, trois boutons et le programme choisi est là, avec la puissance et la tonalité voulues.

Dix manettes, quinze réglages et le circuit oscillateur est accordé, encore quelques mises au point et l'ampli B.F. excite correctement la membrane du haut parleur.

Ces postes de radio sont l'image de deux options importantes. L'un est un produit de consommation, une "machine" qui répond à un besoin qui résoud un problème. L'autre est une "machine" intéressante en soi, et qui, presque par hasard, sert aussi à quelque chose.

Les premiers ordinateurs étaient des machines à calculer dotées de quelques éléments de commande. Ils étaient programmables et pouvaient exécuter tout ce qu'on voulait. Leur langage machine très simple découlait des possibilités du hardware.
"Avec l'instruction de décalage et le branchement conditionnel, vous pouvez tout faire .... " devait être une des réponses classiques à ceux qui doutaient de la puissance et de l'universalité de ces machines.

Et de fait, puisqu'on pouvait tout faire ... on a essayé. Très vite, des outils software se sont imposés. Il fallait un intermédiaire entre ce hardware qui pouvait tout faire et l'utilisateur avec son problème bien précis. Cet intermédiaire, de simple assembleur est devenu un jeu de compilateurs insérés dans un operating system .

Parallèlement à cette évolution du software, les constructeurs d'ordinateurs ont orienté leurs machines en fonction d'une classe de problèmes (par exemple, ceux qui peuvent assez facilement être écrits en COBOL). Ainsi certains langages de Base reflètent assez fidèlement les principaux verbes COBOL ou/et donnent une structure "ALGOL-like" avec les primitives nécessaires aux nouveaux Operating Systems .

Cette évolution vers un hardware de plus en plus spécialisé n'était guère compatible avec une technologie de plus en plus standardisée dans ses fonctions logiques. Le seul élément hardware uniforme dans sa structure (donc économique) et diversifié dans son contenu est la mémoire. Un programme enregistré remplacera un automate cablé dans la commande des différentes unités hardware.

CHAPITRE I

DEFINITION DU HARDWARE.

I. 1. INTRODUCTION AUX PROBLEMES ET METHODES.

La description d'un ordinateur doit remplir 2 fonctions :

D'une part faire comprendre les mécanismes de base de la machine et donner une idée générale à celui qui la découvre pour la première fois. D'autre part être un point de référence précis et non ambigu pour ceux qui soit la construisent, soit l'utilisent.

I.1.1. Méthode et Description d'un ensemble hardware.

Ces 2 fonctions demandent des qualités malheureusement contradictoires. Aussi avons nous choisi de donner 2 descriptions de notre EPRON.

L'une informelle et essentielle, l'autre formelle et fonctionnelle. La première est incomplète, volontairement pour ne pas noyer l'essentiel dans le détail. La seconde lui sert de complément.

Une approche complète de l'EPRON se fera en lisant dans une première phase la description informelle, dans une seconde la description formelle et enfin dans une troisième phase les deux descriptions en parallèle.

Le test de la connaissance de l'EPRON se fera en :

- écrivant un micro-programme de conversion binaire - BCD (pour les lecteurs d'orientation mathématiques ...),
- ou un micro-programme gérant les I/O sur un disque.
- vérifiant ce que l'on a écrit grâce au simulateur.

I.1.2. Langage de description d'un ensemble hardware.

Plusieurs écoles ont chacune fait l'inventaire des langages existants, critiqué l'un ou l'autre aspect puis défini un nouveau langage palliant ces différents maux... Il serait prétentieux de vouloir en quelques lignes suivre leur démarche, et pourtant, pris au jeu, nous avons réalisé non pas un nouveau langage mais un compromis basé sur une "convention des paresseux" quant aux notations utilisées. (1)

REMARQUE.

Ces 2 définitions s'appuient sur un hardware bien précis utilisant la technologie TTL et les systèmes LSI. Une description du hardware en CASSANDRE est en préparation. Cette troisième description sera un complément intéressant pour comprendre le "comment" de la machine, elle n'est cependant pas nécessaire actuellement pour définir nos micro-programmes.

(1) Célèbre convention adoptée par le Professeur Jacques Mersch et stipulant "Ce que j'écris a un sens précis et non ambigu; le lecteur est prié d'ajouter ou de corriger la lettre pour qu'elle corresponde au sens et à la cohérence que je veux lui donner , de même il explicitera les conventions assurant la cohérence de ce texte".

Cette convention inapplicable si la description doit être traitée sur machine est cependant bien utile pour faciliter l'écriture et la compréhension d'un mécanisme par ailleurs défini correctement . (1)

Notre description se fera dans un langage empruntant les notations et la sémantique (nous ne pouvons plus parler de la syntaxe) aux langages définis par messieurs CHU, Mermet et la C.H.B. Quelques notations moins immédiates sont rappelées ci-dessous.

I.1.2.1. Spécification des registres.

- la zone concernée d'un registre est définie par deux nombres, le premier donnant la position du premier bit de la zone, le second indiquant la longueur de la zone;
- si le second nombre est égal à 1, il peut être omis;
- un banc de registre peut être défini en écrivant un indice après le nom du registre.

  Ex. : $(A)17 : 8 \;:=\; (MP_j)1 : 4, 0000$

  la zone de 8 bits de A commençant au bit n° 17 prend la valeur de la zone de mémoire MP (d'adresse j) de longueur 4 et commençant au bit 1, concaténée avec 4 "0".

I.1.2.2. Variables implicites.

- le signe "$*$ " peut remplacer une variable si il n y a pas de confusion possible quant à l'identification de la variable remplacée.

  Ex. : $(CC)1 : 8 \;:=\; * + 1$

  est identique à $(CC)1 : 8 \;:=\; (CC)1 : 8 + 1$

I.1.2.3. Groupe d'instructions-labels.

- un ensemble d'instructions peut être groupé en une entité par l'introduction d'accolades;
- cette entité peut être nommée par un label et exécutée par les instructions PERFORM et EXECUTE.

---

(1) Notre description a été rédigée parallèlement et en référence à un simulateur écrit en Fortran avec routines en langage de base.
Ce simulateur est la définition ultime du fonctionnement de notre machine .
C'est lui qui sert d'interface entre le hardware et le firmware.

c. Détermination de la liste et de l'ordre des paramètres.

Ces paramètres doivent s'incrire dans une structure précise pour qu'à l'utilisation de la Macro ils correspondent à la définition.

0.5.3.2. Opérations exécutées à l'expansion de la Macro.

Le travail se réduit ici à :

a. générer le code Macro adéquat (éventuellement le gérer).

b. lister et générer les paramètres.

0.5.4. Points Précis d'Implantation d'un tel processus.

L'implémentation donnée ici est schématique et fait appel aux notions fondamentales du hard-firm et software plutôt qu'à des machines et packages précisés.

0.5.4.1. Assemblage des micro-programmes.

Il s'agit bien en effet, d'assembler les "corps" de micro-programme de chacune des instructions. Pour ce faire, le Macro-générateur doit avoir à sa disposition une liste complète de ces "corps". Chacun de ces "corps" doit avoir été soigneusement construit pour pouvoir s'imbriquer dans n'importe quelle séquence. Les corps faisant appel à la Mémoire seront facilement paramétrables (pour supporter un paramètre de la MACRO comme argument).
Le Macro-générateur créera donc une Macro en chaînant un en-tête avec une suite de "corps", suivis d'un lien.
Il donne une adresse à cette Macro.

0.5.4.2. Gestion des Codes Macro-Instructions.

Plusieurs méthodes peuvent être utilisées, chacune optimisant un aspect particulier. La solution est peut-être un heureux mélange de deux ou plusieurs méthodes.

a. Code étendu direct.

Un code Instruction est dit "Type Macro". Un code complémentaire est adjoint à ce code pour déterminer la Macro utilisée. Le code complémentaire est suffisamment étendu que pour couvrir toute la liste des Macros pouvant se trouver simultanément dans un programme.

b. Code étendu indirect.

Un nombre n de codes Instructions sont des entrées d'une table donnant les codes complets des Macros accessibles à un instant donné. L'optimisation de la gestion de cette table est complexe et peut être confiée au ..... programmeur ou à un Macro Assembleur évolué.

c. Code restreint..

Un nombre limité de Macros est laissé au programmeur. A chaque Macro, correspond un code Instruction classique.

0.5.5. Conclusions et Perspectives.

Un tel Macro-générateur tout en gardant la même "visibilité" ou programmeur permet d'accroître l'efficience du calculateur micro-programmé.

D'autre part, si dans le jeu des Instructions de Base sont inclues quelques primitives plus "fines", le programmeur a la possibilité de se définir, d'une manière souple, de nouvelles Instructions (Macro-Instructions).

Ce mécanisme étant ascendant (et non descendant, comme pour les compilateurs), l'implémentation, rendue par ailleurs plus simple, peut être optimisée au mieux.

Le langage de définition des Macros étant le langage d'Assemblage, nous réalisons un double objectif :

- faciliter la tâche du programmeur en ne lui imposant pas un nouveau langage pour définir ses Instructions.
- simplifier le "compilateur" en le transformant en un micro-assembleur de micro-séquences préfabriquées.

Parallèlement à la recherche plus théorique, une part importante du travail a été consacrée à définir jusque dans le détail le hardware du projet et à tester son efficacité en écrivant des micro-programmes généraux ou propres à un traitement bien spécialisé.

Il est encore trop tôt pour tirer des conclusions définitives. Cependant jusqu'à présent nous avons pu résoudre les problèmes posés (raccordement de périphériques rapides, protection mémoire, instruction spécialisée, etc..) avec une efficience assez remarquable.
L'étape importante qu'il nous reste à franchir est celle de la définition complète de l'architecture software et un essai de réalisation de système.

L'idée de base (dynamisme dans la relation firmware-software) sous-tend les trois chapitres suivants. Plus particulièrement, nous verrons l'importance accordée aux 2 niveaux de micro-et mini-langage, le premier assurant les primitives de traitement et le second l'enchaînement de ces primitives pour former des instructions.

Le chapitre I présente le hardware qui supportera notre système.

Le chapitre II donne les bases de l'architecture firmware-software.

Le chapitre III enfin propose un mécanisme d'interaction entre software et firmware.

CHAPITRE I

DEFINITION DU HARDWARE.

1. INTRODUCTION AUX PROBLEMES ET METHODES.

1.1. Méthode de Description d'un ensemble hardware.

1.2. Langage de Description d'un ensemble hardware.

1.2.1. Spécification des registres.
1.2.2. Variables implicites.
1.2.3. Groupe d'instructions - label.
1.2.4. Exécution sélective.
1.2.5. Expression logique.
1.2.6. Constantes.
1.2.7. Variables minuscules.

2. DEFINITION FONCTIONNELLE DE L'EPRON.

2.1. Niveaux et enchaînement.

2.1.1. Première approche essentielle.
2.1.2. Seconde approche plus redondante.

2.2. Eléments de l'EPRON et noms des actions.

2.3. Définition formelle de l'EPRON.

2.3.1. Prise en charge d'une nouvelle micro-fonction.
2.3.1. Principaux "sous-programmes".
2.3.3. Opérateur.

2.3.4.1. Champ 1 - Sorties.
2.3.4.2. Champ 2 - Entrées.
2 3.4.3. Champ 3 - Varia
2.3.4.4. Champ 4 - Tests.
2.3.4.5. Champ 5 - Ordres.
2.3.4.6. Champ 6 - Transfert.

2.3.5. Micro câblées et Boutons poussoirs.

2.4. Description du Simulateur.

2.4.1. Description et Fonctionnement.
2.4.2. Langage de Commande.
2.4.3. Mécanisme spécial d'arrêt.

3. DESCRIPTION INFORMELLE DE L'EPRON.

3..1. Schéma général.

3.2. Composants détaillés.

3.2.1. Mémoire Principale.
3.2.2. Bus et Opérateur de transfert.

3.2.3. Registres.
3.2.4. Opérateur.
3.2.5. Système de décode.
3.2.6. Compteurs et clocks.
3.2.7. Système d'interruption et d'I/O.
3.2.7.1. Description.
3.2.7.2. Fonctionnement.

3.3. Commandes.

3.3.1. Tests.
3.3.2. Ordres.
3.3.3. Entrées-Sorties des registres et bascules.

3.4. Rappel de notation pour la micro-programmation.

3.5. Définition de TO.

3.5.1. TON.
3.5.2. TOI.
3.5.3. TOB.

3.6. Définition de st0 et st8.

La micro-programmation était née ... Une succession de mots de contrôle commanderaient les changements d'état de la machine. Avec un hardware simple on pouvait désormais interpréter un langage de base complexe, un décor virtuel plus proche de nos structures de pensée.

Cette notion d'interpréteur a fait son chemin et le langage micro-programme conçu au départ comme une aide dans la réalisation du hardware est devenu un outil au service du système tout entier. Le niveau micro-programme revit actuellement l'évolution vécue jadis au niveau programme. De même que le passage des tabulatrices aux ordinateurs fut marqué par le passage d'une logique câblée à une logique enregistrée, ainsi, une nouvelle gamme de machines se caractérise actuellement, entr'autres améliorations, par un niveau micro-programme développé et adaptable.
La ou les mémoires de contrôle de ces nouveaux ordinateurs sont inscriptibles et les constructeurs proposent même parfois des assembleurs et simulateurs de micro-programme. Certains systèmes, utilisant les techniques d'overlay micro-programmes, présentent même des packages micro-programmes spécialisés dans l'interprétation d'un langage de base propre à une classe d'applications.

En comparant les différents langages de Base depuis les débuts des ordinateurs jusqu'à cette décennie, un jeune informaticien, non-conscient des mécanismes de l'évolution, serait tenté d'employer le mot MACROPROGRAMMATION à propos des nouveaux langages de Base au lieu du terme MICROPROGRAMMATION à propos des langages plus proches du hardware.
Les premiers langages machines sont en effet beaucoup plus proches (fonctionnellement) des langes de micro-programmation actuels que des nouveaux langages de Base (1).

Trois points dans cette histoire guideront notre réflexion :

1. La logique de croissance est de complexifier un niveau jusqu'au seuil de saturation. Ce seuil atteint, un niveau supplémentaire vient dédoubler l'ancien (2).
2. L'architecture actuelle des ordinateurs est basée sur trois niveaux (3) : hardware, firmware, software.
3. L'interaction entre niveaux, laissée aux constructeurs, est actuellement très limitée (4).

(1) BURROUGHS appelle d'ailleurs ses langages de Base des S-Langages; S pour Software, Système, Spécial.

(2) Cette logique de croissance utilisée tous les jours en programmation (routine, programme, chaîne de programmes) se retrouve aussi dans la vie de tous les jours, en manutention, par exemple (ravier, boîte, caisse, containers).

(3) Le QM1, ordinateur expérimental construit aux USA propose un éclatement en deux du niveau firmware, donnant ainsi une architecture à 4 niveaux.

(4) BURROUGHS et HEWLET PACKARD semblent actuellement les pionniers dans cette ligne de recherche.

0. 2. PROSPECTIVES.

Vue sous cet angle, la recherche actuelle prend tout son sens. Si les techniques de micro-programmation sont un mécanisme software permettant de découper la réalisation d'une tâche en primitives de traitement (les instructions du langage de Base), alors, la définition de ces primitives doit tenir compte aussi de la tâche à réaliser, et plus seulement du hardware capable de les supporter.

Une interaction firmware-software pose deux problèmes distincts :

1. Comment définir une architecture hardware-firmware permettant un découpage souple et efficient en primitives de traitement ? Quelles en sont les contraintes ? Sur quelles bases choisir un langage de micro-programmation ?

2. Quel est le type de mécanisme capable d'assurer cette répartition optimale entre software et firmware ? Simple échange d'informations entre programmeurs et micro-programmeurs chez le constructeur, ou, compilateur et macro-assembleur de micro-programmes ?

En recherchant une réalisation concrète, le projet EPRON (Experimental PROcessor Namur) affronte ces problèmes jusque dans le détail. Son objectif est de présenter un ensemble software firmware interactif. Cet ensemble, basé sur un hardware bien précis est réalisé dans le cadre de la construction d'un mini-ordinateur (1) capable de traiter des problèmes de gestion aussi bien qu'un contrôle de processus industriel. Moyennant un interface très réduit (sans nécessité de contrôleurs coûteux) ce mini peut être connecté à chacun des éléments de la gamme classique des périphériques, bandes et disques compris, et traiter ceux-ci en parallélisme avec l'exécution de programmes.

L'architecture de l'EPRON présente une option bien précise parmi une gamme très large de techniques de micro-programmation. Une brève étude de ces techniques permettra de juger plus facilement ce choix et situera l'EPRON dans le courant actuel.

---

(1). La définition d'un mini-ordinateur est basée sur des critères de prix et d'encombrement. D'après Albert B. Tonik, ingénieur conseil chez UNIVAC, un petit ordinateur micro-programmé se caractérise par une mémoire de contrôle de 50.000 bits. Par ailleurs, la majorité des minis peuvent dans leur configuration de base être logés dans un seul rack.
L'EPRON dispose de deux mémoires de contrôle totalisant 40.000 bits dans une première configuration et jusqu'à dix fois plus dans des configurations et applications spéciales.
En configuration de base, un seul rack lui suffit et il se classe (coût-encombrement) dans la gamme des minis.

0. 3. LES GRANDS AXES DE LA MICRO-PROGRAMMATION.

Les différents langages micro-programmes sont constitués de mots de contrôle de longueur fixe, exécutés en un temps court (80 à 500 ns).
Ces mots de contrôle divisés en "champs" déterminent le changement d'état de la machine durant un ou plusieurs temps élémentaires.

Les trois caractéristiques suivantes permettent de classer les différents types de micro-programmation actuellement disponibles.

0. 3.1. Contrôle de l'adresse.

Le mécanisme de prise en charge d'un mot de contrôle détermine fortement les caractéristiques du niveau micro-programme, entr'autres :

- la vitesse d'exécution : look ahead, possible dans le cas d'exécution purement séquentielle.
- le volume de la mémoire de contrôle adressable et la longueur du mot de contrôle : un champ adresse est coûteux en bits.
- le type de branchements possibles et donc la richesse de la structure d'un micro-programme.

Le compromis entre vitesse, volume et richesse de la structure vient du choix fait parmi les techniques ci-dessous pour déterminer l'adresse du prochain mot de contrôle. Cette adresse peut dépendre :

- de l'adresse courante : exécution en séquence.
- du champ adresse du mot de contrôle : branchement explicite.
- de bits caractéristiques d'état : branchements conditionnels.
- de registres spécialisés : pagination, segmentation.

Le mécanisme d'adressage de la plupart des machines actuelles se fonde sur plusieurs (voire même 4) de ces techniques.

0. 3.2. Décode en champs.

Un mot de contrôle définit le changement d'état de la machine pendant un temps élémentaire. Il doit donc commander un certain nombre d'actions élémentaires (ouverture de portes, sélection de registres, positionnement de bascules etc...).
Deux conceptions extrêmes peuvent être envisagées :

1. - à chaque action élémentaire du hardware correspond un bit dans le mot de contrôle. L'action est exécutée si le bit est à 1.

2. - l'ensemble des combinaisons "utiles" des actions est énuméré. Chaque combinaison est codée et détermine un des types de mots de contrôle possibles.

Dans la première conception, la longueur du mot de contrôle est importante, mais par contre la décode nulle est très rapide. La seconde permet un compactage beaucoup plus grand, la mémoire de contrôle a donc des mots beaucoup plus courts, mais le système de décode est plus coûteux en temps et en hardware.
La première conception de plus, n'a pas déterminé à priori toutes les combinaisons utiles; Elle laisse toutes les possibilités du hardware disponibles.
Entre ces deux conceptions existe un compromis basé sur la notion de "champ".
Un champ contrôle un sous ensemble du hardware dont les actions de commande sont mutuellement exclusives. A chacun des éléments principaux du hardware de la machine correspondra un champ logique.
La distinction champ logique - champ physique vient de ce qu'un même champ physique (i.e. une zone du mot de contrôle) peut déterminer plusieurs champs logiques. Un code dans le mot de contrôle précisera pour chaque champ physique le champ logique concerné. (1)
Suivant qu'il y a ou non dans une machine équivalence entre champ physique et champ logique, on dira qu'elle est à champs indépendants ou dépendants. Les remarques énoncées plus haut (bits indépendants ou code général) s'appliquent ici également.

0. 3.3. Indépendance des mots de contrôle successifs.

Les différents composants d'une machine se répartissent grossièrement en 3 classes caractérisées par un temps de base. En unité de temps, ces caractéristiques sont :

Accès mémoire central 100.

Accès mémoire de contrôle et mémoire locale 10.

Passage de l'information à travers une couche logique 1.

La synchronisation entre ces composants se réalise suivant trois techniques (combinaisons de 2 variables hardware-firmware):

a. Synchronisme laissé au micro-programmeur. Celui dispose des différents temps et inclut dans une séquence des mots de contrôle vierges (= n'utilisant pas le composant à synchroniser) (2).

---

(1) La décode se réalise donc en 2 couches de logique :

- sélectionner l'unité hardware concernée par le champs.
- sélectionner l'action déterminée par le contenu du champs.

(2) Exemple : pour commander une lecture en mémoire, le micro-programmeur devra successivement :

- charger le registre adresse
- donner un ordre de lecture à la mémoire.

-} réaliser des actions qui ne concernent pas la mémoire.
-

- sortir la mémoire.

b. Synchronisme laissé au micro-programmeur et supporté par le hardware. Un champ dans le mot de contrôle précise une fonction de retard.

c. Synchronisation hardware. Mode asynchrone synchronisé réalisé par figeage de l'unité la plus rapide défigée par la plus lente ou par mécanisme de coulisses dans l'horloge.

0. 3.4. Lignes de recherches actuelles.

0. 3.4.1. Multiplication des niveaux.

Vu l'évolution du niveau firmware, les langages micro-programmes dans certaines machines deviennent tellement évolués qu'ils peuvent être comparés aux langages de base de jadis.
Il est alors naturel d'envisager un niveau supplémentaire pour interpréter les micro-programmes. Les termes mini-programmation, micro-programmation et pico-programmation devenus maintenant classiques témoignent de cette multiplication des niveaux.
Souvent dans une même machine, les différents niveaux auront des propriétés très différentes et équilibreront à leur façon :

- l'adressage.
- la décode en champs.
- l'indépendance des mots de contrôle successifs.

0. 3.4.2. Dynamisme firmware.

Plusieurs machines disposent de mémoire de contrôle inscriptible. Cette facilité est utilisée durant la phase de mise au point des micro-programmes ou comme zone d'overlay pour des micro-programmes spécialisés. Certaines machines banalisent leurs mémoires au niveau architecture. Le software et le firmware partagent alors la même mémoire. Cette mémoire unique peut être divisée en blocs physiques réalisés dans des technologies différentes et redonner un temps d'accès réduit pour la zone de base du firmware et du software système.

0. 3.4.3. Aide à la réalisation de micro-programmes.

Les constructeurs mettent de plus en plus à la disposition du client (pour les minis) des assembleurs et simulateurs de micro-programmes. Ces programmes sont même conçus pour "tourner" sur une configuration moyenne du mini. On remarque cependant que les minis qui donnent ces possibilités ont un langage de micro-programmation assez figé; le nombre de champs dans un mot de contrôle est réduit et la dépendance entre champs est grande. On peut dire qu'ils donnent accès au mini-langage, et non au micro-langage.

0.4. L'EPRON face à ces axes.

L'interaction firmware-software, objectif du projet EPRON, implique la possibilité de découper une tâche globale en une séquence d'unités de traitement dites primitives de la tâche. Le firmware met à la disposition du software un jeu de séquences de telles primitives. L'interaction firmware-software permet de modifier ces primitves et leur type de chaînage en fonction de la tâche globale à exécuter.

Pour supporter facilement un tel découpage - assemblage modifiable par le software, le système fimware-hardware est bâti sur une structure à deux niveaux de langage micro-programme.
Le micro-langage permet de définir et d'exécuter des unités de traitement, le mini-langage les chaîne pour assurer l'interprétation des instructions. Tous deux sont en mémoire vive pour assurer leur adaptation par le software.

Cette structure de base définit presqu'automatiquement des critères de sélection dans chacun des domaines précédents (contrôle de l'adresse, décode en champs, indépendance des mots de contrôle).

0. 4.1. Contrôle de l'adresse.

L'adresse du mot de contrôle est obtenue par concaténation de deux registres. Celui de poids fort détermine l'unité de traitement à réaliser, et celui de poids faible les différentes étapes de son exécution. 256 unités de traitement sont ainsi adressables et 8 étapes sont prévues pour chacune d'elles. La mémoire de contrôle rapide contient les 2 K mots de contrôle (256 * 8 = 2 K). La mémoire centrale banalisée software-firmware contient le mini-langage et permet le chaînage des unités de traitement.
Deux types de commandes agissent sur l'adresse de la mémoire rapide de contrôle :
celles internes à l'unité de traitement (saut d'un mot de contrôle, reprise au premier mot de contrôle, fin de l'unité de traitement) et d'autres, externes qui font appel au mini-langage pour préciser la nouvelle unité de traitement à exécuter (1).

Cette technique permet le look-ahead au niveau de l'unité de traitement et optimise au mieux la gestion de l'adresse rendue indépendante du mot de contrôle. En effet, le niveau mini-langage seul supporte le contrôle explicite de l'adresse.

---

(1) Les interruptions sont reprises au niveau, mini-langage, tandis que les tests et sauts conditionnels élémentaires sont effectués au niveau micro-langage.

0. 4.2. Décode en champs.

Les deux niveaux de mini- et micro-langages permettent une grande souplesse dans le chaînage des unités de traitement. Pour garder à ces unités de traitement toute leur puissance et ne pas restreindre à priori leur variété, un encodage par champs s'impose.
Ces champs commandent chacun une unité hardware et l'indépendance est pratiquement totale. De plus, cette option permet de profiter au maximum de la technologie actuelle. Toute configuration de bits dans le mot de contrôle réalise une transformation bien précise de l'état de la machine. Parmi ces configurations nous devrons, en étant sévère, en rejeter 99 % qui ne répondent à aucune fonctionnalité. Malgré ce déchet (1), le résultat global est économique. Le coût de la mémoire rapide pourrait être réduit de 30 % mais il serait contrebalancé par un hardware très alourdi et plus lent.

Le niveau mini-langage étant déjà interprété en partie par le micro-langage, il ne peut être question de champs et de décode hardware.

0. 4.3. Indépendance des mots de contrôle.

La question est délicate et toutes les décisions sont loin d'être prises au niveau hardware. Un point est clairement défini : la visibilité de la machine au niveau du micro-programmeur. A ce niveau, l'indépendance des mots de contrôle est totale. Cette option vient d'une part,de la volonté de faciliter le travail des micro-programmeurs et d'autre part, de permettre une densité maximum dans l'écriture des mots de contrôle (2).

Dans une première phase, le hardware simplifié au maximum ne tient pas compte des optimisations possibles par asynchronisme. Une seconde phase pourra le conduire à augmenter sa rapidité pour peu de frais hardware mais beaucoup de "matière grise".

---

(1) Si 99 % des configurations sont à rejeter, cela veut dire que 7 bits ($2^7 = 128$) au plus sont perdus sur les 24 d'un mots de contrôle soit : $7 : 24 \cong 30$ % de déchet.

(2) Cette densité très élevée explique en partie le résultat satisfaisant de l'encodage par champs. La suppression de cette synchronisation, entraîne immédiatement une perte de plus de 50 %.

Cette seconde phase d'optimisation s'appuie également sur d'autres techniques que l'asynchronisme. Nous en esquissons deux ci-dessous.

A partir de la définition sémantique et des combinaisons de commandes les plus usuelles dans les mots de contrôle, un "poids" peut être donné à chaque unité hardware. (1) Ce "poids" est un critère important dans le choix des composants et de la structure d'implantation.

D'autre part, dans le cas de chaînes critiques (2), un circuit rapide et simple peut détecter "à l'avance" une condition logique et prévenir l'horloge qu'elle devra prolonger son temps élémentaire. Ce système "à coulisses" donne au hardware le temps dont il a besoin plutôt que de lui donner rigidement un temps élémentaire égal au temps de la chaîne critique du système.

0. 5. MACRO-ASSEMBLEUR SUPPORTE PAR FIRMWARE.

Les systèmes actuels donnent des facilités dans le découpage d'une tâche globale en chaîne de programmes, en programmes et en routines ou procédures. Si l'utilisateur veut affiner le traitement, il peut encore descendre jusqu'au langage de base et disposer de l'aide du système pour réaliser les interfaces avec les autres niveaux.

Le niveau micro-programme, lui, est soit inaccessible, soit dépourvu de tout support software pour le relier au reste du système. Pourtant, bénéficier de ce niveau peut s'avérer très économique en place mémoire, et plus efficient en temps d'exécution.

Le macro-assembleur supporté par firmware est un mécanisme accessible au niveau d'un macro-assembleur ou du langage de base. Il donne la possibilité au programmeur de profiter de la puissance du firmware tout en ne lui imposant pas la manipulation de nouveaux langages.

---

(1) Exemple :

Les premières adresses de la mémoire centrale sont beaucoup plus utilisées que celles de poids supérieur. Un hardware optimalisé réaliserait donc dans une technologie différente les positions basses et hautes de la mémoire.
Il ne s'agit pas ici de modifier la structure logique de l'architecture ou sa visibilité firmware mais bien de maximiser un rapport performance prix dans l'implantation hardware.
Autre exemple pris en horlogerie : Les axes du système d'échappement pivotent sur des rubis; ceux qui sont moins sollicités sont directement en contact avec l'alliage du chassis. La fonction de l'axe et de son pivot n'a pas changé mais la réalisation a tenu compte de critères usure/prix.

(2) Une chaîne critique est un chemin de longueur maximale dans le graphe représentatif d'un système d'équations logiques.

0.5.1. Principe.

Le principe très simple repose sur une double constatation :

- une instruction en langage de base est exécutée par un micro-programme dont le début et la fin sont consacrés à des opérations de "fetch" (1).

- dans un programme généré par un compilateur ou un macro-assembleur (2), un grand nombre d'instructions font partie de séquences standards très souvents répétées.

Il s'énonce comme suit :

L'expansion d'une macro en langage de base dans le texte du programme est remplacée par un branchement micro-programme (défini par le code instruction). La définition de la macro provoque l'assemblage d'un microcode correspondant à la sémantique de la macro et l'assignation d'un nouveau code instruction.

0.5.2. Justification.

Un rapide calcul donne les conditions de gain de temps et espace mémoire.

0.5.2.1. Gain de temps.

Soient :

Tc ::= Temps de chaînage ::= Temps machine nécessaire pour effectuer une en-tête et un lien.

Te ::= Temps d'Exécution ::= Temps machine nécessaire pour effectuer les opérations définies par une instruction donnée.

Ti ::= Temps d'Instruction ::= Tc + Te.

Soit un programme donné ; il se caractérise par :

TE ::= Σ Te fixe.

TC ::= Σ Tc à réduire.

TP ::= Temps du Programme ::= TE + TC à optimiser.

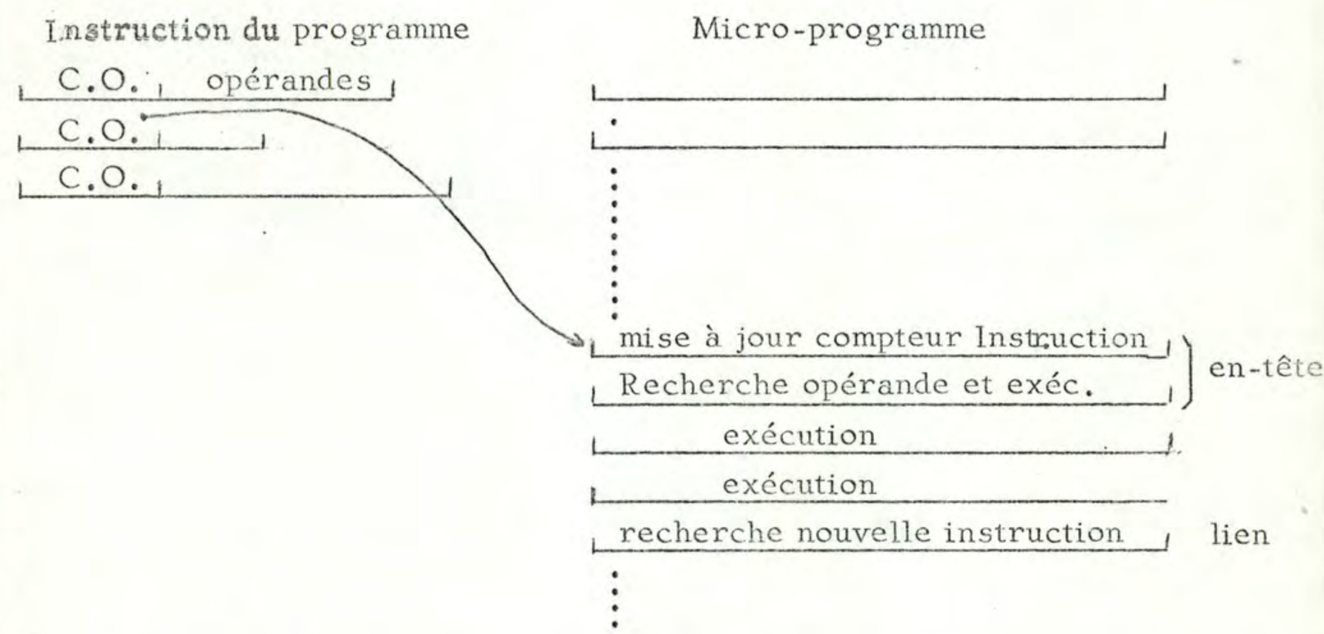
---

(1) La figure ci-contre visualise ce mécanisme.

(2) Les termes "macro-assembleur, macro définition, macro expansion" employés ici font référence au mécanisme de génération de code proprement dit, sans tenir compte de l'assemblage conditionnel et paramétré.

Prise en charge des Instructions dans une machine micro-programmée.

Dans un tel type de machine, les instructions sont, en fait, considérées comme des données par le micro-programme.
La technique de sélection par adressage est très utilisée et en général, à un code instruction correspond une adresse micro-programme. Cette prise en charge d'une instruction est représentée par le diagramme ci-dessous :

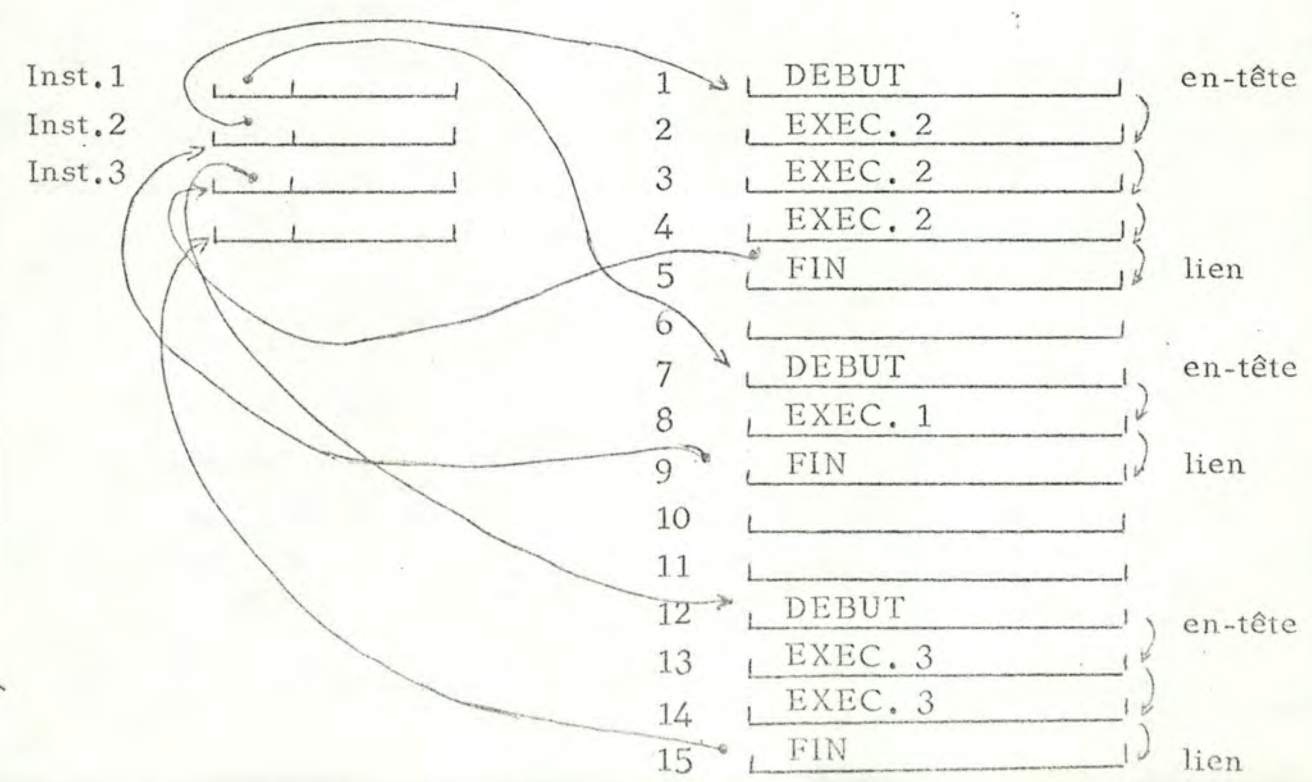


Un en-tête et un lien (fin) entourent donc le micro-programme propre à chaque instruction.

FIGURE

Exécution d'une séquence d'instruction.

Dans le diagramme ci-dessous, les flèches visualisent la circulation de l'information de chaînage. A chaque flèche correspond un adressage Mémoire (Centrale ou de Contrôle).



Réduire Tc équivaut, soit à réduire chacun des Tc, soit à réduire leur nombre. Ce second aspect est traité ici.
Une séquence d'instructions répètée souvent, devient une seule instruction avec 1 seul Tc. Le Temps gagné sera ainsi proportionnel au nombre de fois que la séquence est exécutée, multiplié par le nombre d'instructions de cette séquence.

0.5.2.1. Gain de place.

Soient :

Ps ::= Longueur d'une séquence (en unités Mémoire) écrite dans le Programme.

Ms ::= Longueur d'une séquence (en unités Mémoire) écrite en Micro-programme.

Ns ::= Nombre d'occurrences de la séquence dans le Programme.

Remplacer une séquence d'instruction par une seule instruction en générant un micro-programme spécial donne un gain de place si $N_s * (P_s - 1) > M_s$

$$N_s * (P_s - 1) > M_s$$

N.B. :
On remarquera, au passage, qu'il s'agit bien d'optimiser une fonction complexe.
Compiler tout le programme en 1 seul micro-programme serait assez coûteux en place Mémoire et en temps compilation ! ... bien que le plus efficace en temps d'exécution !

0.5.3. Réalisation.

Le macro-assembleur réalise deux types de fonctions : lors de la définition et lors de l'expansion de la macro. Pour chacune d'elles, le firmware devra effectuer les opérations suivantes :

0.5.3.1. Opérations exécutées à la définition de la Macro.

a. Gestion d'un code Macro.

La macro doit en effet être identifiée pour pouvoir être exécutée. Le code doit pouvoir donner accès au micro-programme qui exécutera cette macro.

b. Génération du micro-programme.

A partir de la définition de la Macro, il faut générer la suite de micro-instructions qui vont effectivement exécuter la sémantique de cette définition.

I.1.2.4. Exécution sélective EXECUTE (parl, labl, ...labn).

- en fonction de la valeur de parl, il y a exécution du "bloc" défini par le ième label où i = valeur de parl + 1.

- est équivalente à l'instruction PERFORM labi avec la facilité de choix du label.

I.1.2.5. Expression logique.

Opérateurs $\overline{\quad}$, $\wedge$, $\vee$, $\veebar$ respectivement, non, et, ou et ou exclusif.

I.1.2.6. Constantes.

Sont définies en digits binaires, octaux, décimaux, hexadécimaux, ou en nombres décimaux. La convention des paresseux permet de choisir la forme voulue.

I.1.2.7. Variables minuscules.

- sans faire référence explicitement à un élément hardware elles permettent des simplifications d écritures et rendent la compréhension plus facile.

I. 2. DEFINITION FONCTIONNELLE DE L'EPRON.

Cette définition est bien celle d'un modèle à l'intention du micro-programmeur. Celui-ci a en face de lui un automate qu'il devra mener d'un état i jusqu'à un état i + k, correspondant à l'exécution d'une instruction.

Pour faciliter la compréhension de cet automate, nous indiquons rapidement quelques structures de notre architecture ainsi que les noms que nous leur donnons.

I.2.1. Niveaux et enchaînement.

L'architecture de l'EPRON est basée sur une micro-programmation à 2 niveaux, tous deux en mémoire inscriptible. Cette décomposition en niveaux permet de définir des unités élémentaires de traitement, de leur donner un nom (i.e. un numéro) et de pouvoir ensuite, au niveau supérieur, les utiliser comme primitives dans un traitement plus complexe.

I.2.1.1. Première approche essentielle.

Partant du plus élémentaire, nous avons dans l'EPRON, les entités suivantes.

1. Actions de 6 Types :

   1. Sortie d'un registre sur les bus S.
   2. Entrée dans un registre à partir des bus E.
   3. Opérateur-Varia - type d'opération.
   4. Tests.
   5. Ordres.
   6. Mode de transfert des bus S aux bus E.

   - Chaque Type comprend 16 actions possibles.
   - A chaque Type correspond 1 champ de 4 bits.
   - Ce champ permet de coder l'action choisie parmi les 16.

2. Micro-Fonction composée de 6 actions, une de chaque Type :

   - est définie par 24 bits (= 6 x 4) groupés en 6 champs.
   - est exécutée par le hardware en un sous-temps élémentaire (250 ns).

3. Micro-Instruction composée de 8 Micro-Fonctions.

   - est nommable par un numéro (8 bits) donnant l'adresse en Mémoire Rapide de la séquence des 8 Micro-Fonctions la composant,
   - constitue l'unité au niveau Mini-langage.
   - est exécutée en un Temps de 9 sous-temps st0=Fetch, st1... st8

4. Ligne Micro composée de 3 Micro-instructions.

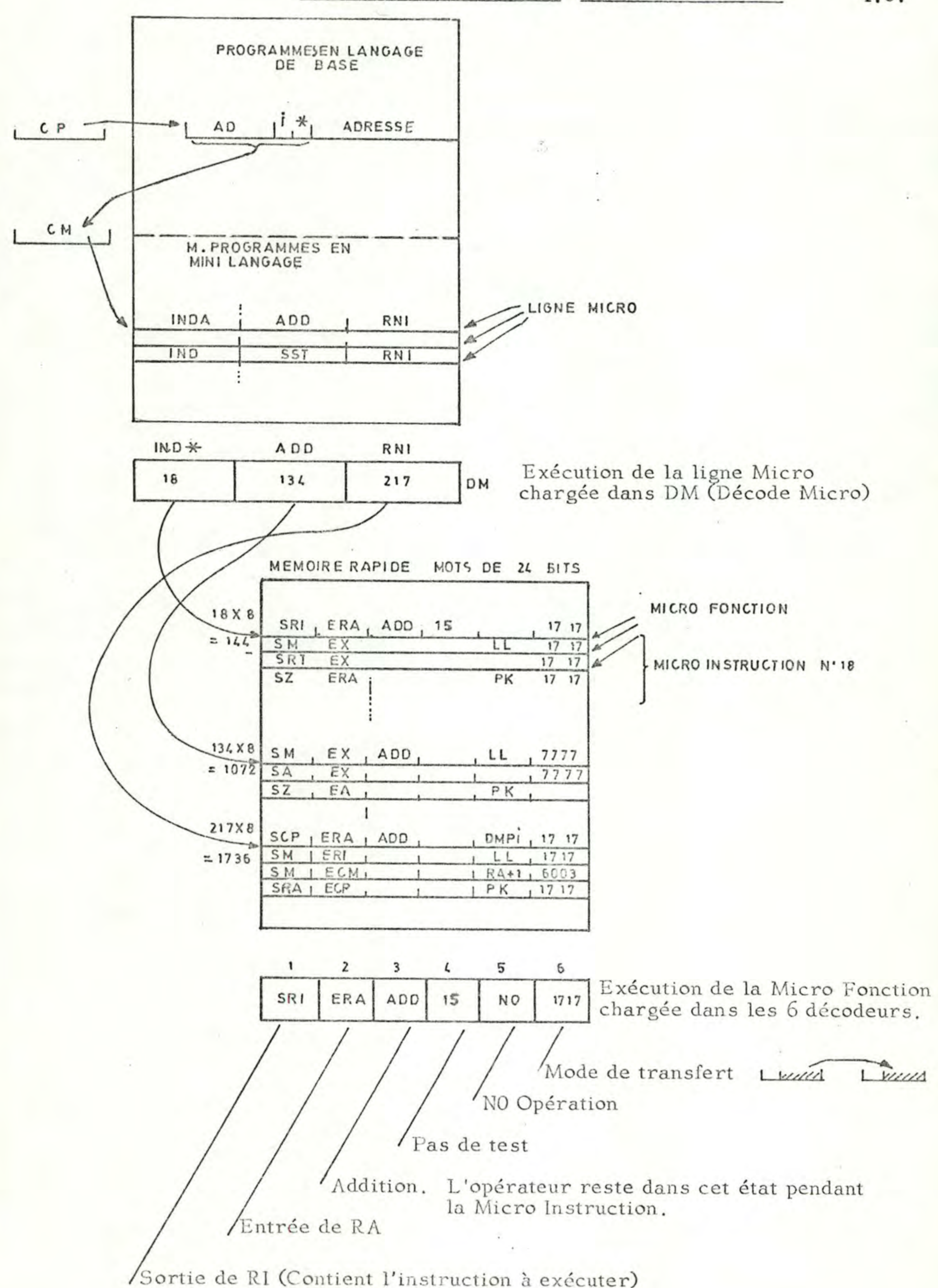
   - est nommable par son adresse en Mémoire Principale,
   - est définie par 24 bits (= 3 x 8)
   - est exécutée en 4 Temps T0-Fetch, T1, T2, T3,

5. Instruction composée d'un nombre variable de Ligne Micro

   - est nommable par son code Opération
   - est définie par une suite de Lignes Micro.

   N.B. : Ce niveau étant la limite entre Software et Firmware, aucune précision ne peut être donnée indépendamment d'un langage de Base préalablement défini.

La figure à la page suivante permettra de mieux visualiser l'enchaînement des différents niveaux.


MEMOIRE PRINCIPALE MOTS DE 24 BITS
PROGRAMMES EN LANGAGE DE BASE
CP
AD
i
*
ADRESSE
CM
M.PROGRAMMES EN MINI LANGAGE
INDA
ADD
RNI
IND
SST
RNI
LIGNE MICRO
IND *
ADD
RNI
18
134
217
DM
Exécution de la ligne Micro chargée dans DM (Décode Micro)
MEMOIRE RAPIDE MOTS DE 24 BITS
18 X 8 = 144
SRI ERA ADD 15 17 17
SM EX LL 17 17
SRT EX 17 17
SZ ERA PK 17 17
MICRO FONCTION
MICRO INSTRUCTION N° 18
134 X 8 = 1072
SM EX ADD LL 7777
SA EX 7777
SZ EA PK
217X8 = 1736
SCP ERA ADD DMPi 17 17
SM ERI LL 1717
SM ECM RA+1 6003
SRA ECP PK 1717
1 2 3 4 5 6
SRI
ERA
ADD
15
NO
1717
Exécution de la Micro Fonction chargée dans les 6 décodeurs.
Mode de transfert
NO Opération
Pas de test
Addition. L'opérateur reste dans cet état pendant la Micro Instruction.
Entrée de RA
Sortie de RI (Contient l'instruction à exécuter)

I.2.1.2. Seconde approche plus redondante.

Vu l'importance de ces notions et de leurs noms, nous nous permettons de redéfinir sous une autre forme les 4 unités de traitement.

Types d'actions :

15 Sorties
15 Entrées
15 Tests
16 Ordres
16 Modes de Transfert
16 Types d'Opération
15 Actions complémentaires.

Une action définit soit un niveau logique pendant 1 sous-temps (ex. SA fixe la valeur des Bus de Sortie à celle de (A)) soit, une impulsion positionnant une bascule (ex. OR remet le Report à 0).

Une micro-fonction ou mot de commande est un mot de 24 bits découpé en 6 champs. Chacun de ces champs définit une et une seule action. Une micro-fonction définit entièrement la transformation de l'état de la machine pendant 1 sous-temps.

champ 1	2	3	4	5	6
Sortie	Entrée		Tests	Ordres	Transfert

Une micro-instruction ou séquence de micro-fonctions est une séquence de 8 mots de Mémoire Rapide. Cette séquence est adressée d'une part,par le code de la micro (8 bits) et d'autre part, par le compteur de sous-temps pour les 3 bits de poids faible.

Compteur s.t.

Code Micro
8 bits

micro 63	s.t.7
micro 63	s.t.8
micro 64	s.t. 1
micro 64	s.t. 2
micro 64	s.t. 3
micro 64	s.t. 4
	:
	:

Une ligne micro ou mini-instruction est un mot de 24 bits découpé en 3 octets définissant chacun une micro-instruction (8 bits).
Le passage d'un octet à l'autre(i.e. d'une micro à l'autre) se fait sous contrôle du compteur de Temps.

micro 1	micro 2	micro 3
Temps 1	Temps 2	Temps 3

Le temps 0(T0) définit la prise en charge d'une nouvelle ligne micro.

Une ligne micro est donc faite en 3 Temps ; chacun de ces Temps se déroulant en 8 sous temps correspondant à l'exécution d'une micro-instruction, c'est-à-dire d'une séquence de 8 micro-fonctions.

I.2.2. Eléments de l'EPRON et noms des actions.

Les différents éléments hardware visibles du micro-programmeur sont énoncés ci-dessous. Ils entreront dans la définition formelle donnée plus loin.

Élément	Désignation
(A)1:24	Registre Scrat
(B)1:24	Registre Scrat
(RT)1:24	Registre Scrat
(RI)1:24	Registre Scrat
(CM)1:24	Registre Scrat
(CP)1:24	Registre Scrat
(R1)1:24	Registre Scrat
(R2)1:24	Registre Scrat
(X)1:24	Opérande 1
(Y)1:24	Opérande 2
(Z)1:24	Résultat
R	Report Initial
D	Débordement
(RA)1:24	Registre Adresse Mémoire
(SM)1:24	Sortie Mémoire
BAMP	Bit Adresse Mémoire Principale
(RB)1:255	Request Bank
(RAP)1:8	Registre Adresse Périphérique
MPI	Masque Priorités Inférieures
(BUS EXT)1:24	Bus Extérieur

Élément	Désignation
(R8)1:8	Registre de 8 bits indépendants
(DM)1:24	Décode Micro
(CC)1:8	Compteur de cycles
BAMR	Bit adresse Mémoire Rapide

(AC)1:24 : := (RAP)1:8, xxxx xxxx, (CC)1:8

(A8C)1:24 : := (RAP)1:8, (R8)1:8, (CC)1:8

Élément	Désignation
(MP_i)1:24	$0 \leqslant i \leqslant 65535$ Mémoire Principale
(MR_i)1:24	$0 \leqslant i \leqslant 2047$ Mémoire Rapide
(CT)1:2	Compteur de Temps
(CST)1:4	Compteur de sous-temps
(RC)1:24	Registre Commutateurs
(COM)1:3	Commutateur
IEDM	Switch Inhibe Entrée DM

Ces trois derniers éléments sont commandés au panneau avant.

Le tableau ci-dessous donne la liste des actions que peut écrire le micro-programmeur.

Elles sont classées en 6 champs.

	1	2	3		4	5	6		
	Sorties	Entrées	Opér.	Varia	Tests	Ordres	TRSF	ER8	
0	NO	NO	ADD1	NO	1S	NO	7777	1R81	0
1	SA	EA	NOT	NO	1R8S	K	1460	2R81	1
2	SB	EB	SP1	EX	2R8S	SEQ	0360	3R81	2
3	SRT	ERT	NOR	IAMP	3R8S	SMF	6003	4R81	3
4	SRI	ERI	SST	IAMR	4R8S	TZ	1403	5R81	4
5	SCM	ECM	EX	IMPI	5R8S	1R	0303	6R81	5
6	SCP	ECP	SP2	OAMP	6R8S	OR	0101	7R81	6
7	SR1	ER1	ZERO	OAMR	7R8S	LL	0201	8R81	7
8	SR2	ER2	SHFT	OMPI	8R8S	LE	7007	1R8R	8
9	SZ	EX	SP3	NO	1XS	RA+1	0707	2R8R	9
A	SRA	EY	ADD	NO	ELS	I	1717	3R8R	A
B	SM	EDM	SP4	NO	DS	DMPI	3776	4R8R	B
C	SRC	ERA	SP5	NO	1XR	F	7637	5R8R	C
D	SDM	EAC	OR	NO	DR	RP	0102	6R8R	D
E	SA&C	ER8	SST1	NO	TZOS	CLR8	0314	7R8R	E
F	VE	VS	AND	I	DPK	PK	6060	8R8R	F

TABLEAU DES ACTIONS

Signification des variables minuscules intervenant dans la définition formelle de l'EPRON.

ad	adresse::= contenu de (RA)9 : 16.
ad1	adresse 1 := adresse corrigée par BAMP ou BAMR et servant soit à la mémoire principale, soit à la mémoire rapide.
ap	adresse priorité := résultat de l'encodage des requests.
apr	adresse priorité := request = résultat de l'encodage des requests en tenant compte du masque.
bcc	bit compteur de cycle : permet l'inhibition de l'effet de CC dans la micro(ligne micro) où il a été chargé.
blmic	bouclage ligne micro : résume les conditions de bouclage sur une ligne micro.
bmicro	bouclage micro instruction : idem blmic sur une micro.
bs	bascule saut : mémorise au cours d'un sous-temps la nécessité de saut du sous-temps suivant.
(e)1:24	Bus E.
fig	figeage : mémorise au cours d'un sous-temps la nécessité de figeage au sous-temps suivant.
i	position du premier bit (défini par le mode de transfert) à charger dans le registre récepteur.
inv	inversion : signale la présence d'un I dans les champs 3 ou 5.
j	position du dernier bit (défini par le mode de transfert) à charger dans le registre récepteur.
k	clock : mémorise l'action à réaliser sur l'horloge.
l	numéro du bit de R8 à charger.
(op) 1:4	opérateur : mémorise le code opération défini par le champ 3 aux sous-temps 1 et 5.
(p)1:8	priorité : mémorise la valeur de (RAP)1:8
pk	"petit"clock : mémorise l'action à réaliser sur l'horloge.
(s) 1:24	Sortie.
seq	séquence : mémorise l'action à réaliser sur l'horloge et sur (RAMR)1:8.

smf sortie micro fonction : mémorise le fait que le sous-temps suivant ne doit pas être exécuté et délivre seulement une constante.

I.2.3. Définition formelle de l'EPRON.

I.2.3.1. Prise en charge d'une nouvelle micro-fonction.

bmicro := $\overline{\text{bcc}} \wedge (\text{CC})1{:}8 \neq 0 \wedge \overline{\text{MCC}}$

blmic := $\overline{\text{bcc}} \wedge (\text{CC})1{:}8 \neq 0 \wedge \text{MCC}$

(CST)1:4 := ж +1

IF bs, (CST)1:4 := ж + 1 — Saut ?

IF pk $\vee$ k $\vee$ séq $\vee$ (CST)1:4 > 1000, — Nouvelle instruction ?

- IF $\overline{\text{seq}} \wedge \overline{\text{bmicro}}$, (CT)1:2 := ж + 1
- IF bmicro, (CC)1:8 := ж - 1
- IF seq, (RAMR)1:8 := ж +1
- IF k $\vee$ (CT)1:2 = 00, — Nouvelle ligne micro ?
  - PERFORM T0
  - IF blmic,
    - (CC)1:8 := ж - 1 — Bouclage sur ligne micro.
    - IF (CC)1:8 = 0, MCC : = False
  - ELSE
    - IF (CC)1:8 = 0 $\wedge$ (apr)1:8 $\neq$ 0,
      - (RAP)1:8 := (apr)1:8 — Interruption
      - (RA)1:24 := 0000 0000 0000 01, (RAP)1:8,00
    - ELSE,
      - (RA)1:24 := (CM)1:24
      - (CM)9:16 := ж + 1 — Prise en charge normale
      - IF COM = 4 $\wedge$ MPI $\wedge$ (apr)1:8 = 0, fig := True
      - MPI : = False
    - (ad)1:16 := (RA)9:16
    - PERFORM Lectmém.
    - IF $\overline{\text{IEDM}}$, (DM)1:24 := (SM)1:24 — Chargement de DM
- IF $\overline{\text{seq}} \wedge \overline{\text{bmicro}}$,
  - IF (CT)1:2 = 01, (RAMR)1:8 := (DM)1:8 — 1er Temps
  - IF (CT)1:2 = 10, (RAMR)1:8 := (DM) 9:8 — 2èm Temps
  - IF (CT)1:2 = 11, (RAMR)1:8 := (DM)17:8 — 3èm Temps
  - IF (RAMR)1:5 = 00000, — Micro cablée ?
    - PERFORM hardmicro
    - GØTØ EXTERNE
  - ~~ELSE,~~
  - PERFORM ST0

```
(RAMR)9 : 3 := (CST)2:3
PERFORM LectmémR
IF COM = 1 ∧(apr)1:8 = 0, fig := True
bs := False
IF smf,                                         Sortie micro fonction ?
   (RT)1:24 := (SMR)1:24
   ELSE
   ⎡ (ad)1:16 := (RA)9 : 16
   ⎢ (p)1:8    := (RAP)1:8
   ⎢ (MF)1:24 := (SMR)1:24
   ⎢ IF (MF)17:4 = LL, PERFORM Lectmém
   ⎢ PERFORM Opérateur
   ⎣ PERFORM Micro Fonction
GØTØ EXTERNE
```

I.2.3.2. Principaux sous-programmes.

```
TO    ⎡ (X)1:24 := 0
      ⎢ IF COM = 3 ∧ (apr)1:8 = 0, fig := True
      ⎣ (CT)1:2 := 1

STO   ⎡ (Y)1:24 := 0
      ⎢  R := 0
      ⎢ smf := False
      ⎢ pk := False
      ⎢ k := False
      ⎢ séq := False
      ⎢ IF COM = 1 ∧(apr)1:8 = 0, fig := True
      ⎢ IF (CC)1:8 ≠ 0 ∧ bcc ∧(MCC V (CT)1:2 ≠ 01),
      ⎢         (CC)1:8 := x - 1
      ⎢         IF (CC)1:8 = 0, MCC := False
      ⎢ IF (CT)1:2 = 01 ∧ MCC, bcc := False
      ⎣ (CST)1:4 := 1
```

Lectmém

(ad1)1:16 := (ad)1:16
IF(ad)1:12 = 000 ∧ BAMP, (ad1)8 := 1
(SM)1:24 := $(MP_{(ad1)1:16})1:24$

Lectmém R

(ad1)1:11 := (RAMR)1:11
IF (ad1)1:5 = 00000 ∧ BAMR, (ad1)5 := 1
(SMR)1:24 := $(MR_{(ad1)1:11})1:24$

Micro-Fonction

```
EXECUTE ( (MF)1:4, SA, SB, SRT, SRI, SCM, SCP, SR1, SR2,
                     SNO, SZ, SRA, SM, SRC, SDM, SA8C, VE)
(e)1:24 := 0
EXECUTE ( (MF)21:4, 7777, 1460, 0360, 6003, 1403, 0303, 0101, 0201,
                     7007, 0707, 1717, 3776, 7637, 0102, 0314, 6060
IF (MF)17:4 = TZ,
      [ i := 1
      [ J := 24
inv := (MF)17:4 =i V(MF)9:4 = i ∧(CST)1:4 ≠ 0001
EXECUTE ((MF)13:4, 1R8S,2R8S,3R8S,4R8S,5R8S,6R8S,7R8S,8R8S
                    1S,  ,1XS, ,ELS ,DS   ,1XR  ,DR   ,TZ0S,DPK
EXECUTE( (MF)17:4, CNO  ,K    ,SEQ ,SMF,TZ   ,1R   ,OR   ,LL,
                    LE   ,RA+1,I    ,DMPI,F   ,RP   ,CLR8,PK )
EXECUTE ( (MF)5:4 , EA   ,EB   ,ERT ,ERI ,ECM ,ECP ,ER1  ,ER2
                    EX   ,EY   ,EDM ,ERA ,EAC ,ER8 ,VS   ,ENO
IF (CST)1:4 = 0001 V x = 0101,
      (op)1:4 := (MF)9:4
      ELSE,
EXECUTE ( (MF)9:4, VNO  ,VNO ,EX   ,1AMP,1AMR ,1MPI ,0AMP,0AMR
                    OMPI ,VNO,VNO  ,VNO,VNO  ,VNO, ,VNO  ,I  )
```

I. 2.3.3. Opérateur.

Opérateur [EXECUTE((OP)1:4, ADD1, NOT, SP1, NOR, SST, EX, SP2, ZERO, SHFT, SP3, ADD, SP4, SP5, OR, SST1, AND)

ADD1 D, (Z)1:24 := (X)1:24 + R

NOT [(Z)1:24 := $(\bar{X})$1:24
D := 0

NOR [(Z)1:24 := $\overline{(X)1:24 \vee (Y)1:24}$
D := 0

SST D, (Z)1:24 := (X)1:24 + $(\bar{Y})$1:24 + R

EX [(Z)1:24 := (X)1:24 $\veebar$ (Y)1:24
D := 0

ZERO D, (Z)1:24 := 0

SHFT D, (Z)1:24 := (X)1:24, R

ADD D, (Z)1:24 := (X)1:24 + (Y)1:24 + R

OR [(Z)1:24 := (X)1:24 $\vee$ (Y)1:24
D := 0

SST1 D, (Z)1:24 := (X)1:24 + FF FF FF + R

AND [(Z)1:24 := (X)1:24 $\wedge$ (Y)1:24
D := 0

SP1
SP2
SP3
SP4
SP5

I. 2.3.4.1. CHAMP 1 Sorties.

```
SA      (s)1:24 := (A)1:24
SB      (s)1:24 := (B)1:24
SRT     (s)1:24 := (RT)1:24
SRI     (s)1:24 := (RI)1:24
SCM     (s)1:24 := (CM)1:24
SCP     (s)1:24 := (CP)1:24
SR1     (s)1:24 := (R1)1:24
SR2     (s)1:24 := (R2)1:24
SNO     (s)1:24 := 0
SZ      (s)1:24 := (Z)1:24
SRA     (s)1:42 := (RA)1:24
SM      (s)1:24 := (SM)1:24
SRC     (s)1:24 := (RC)1:24
SDM     (s)1:24 := (DM)1:24
SA8C    (s)1:24 := (RAP)1:8, (R8)1:8, (CC)1:8,
VE      IF (p)1:8 ≥ 240 ∨ (p)1:8 ≤ 3,
            IF (p)1:8 < (ap)1:8 ∨ (ap)1:8 = 0,
                (ap)1:8 := (p)1:8
                (RB)p := 1
                IF ‾MPI‾ ∧ (ap)1:8 ≥ 240,
                    (apr)1:8 := 0
                    ELSE,
                    (apr)1:8 := (ap)1:8
            ELSE,
            CALL PERIPHE (p)
            IF (RAP)8 = 1, (s)1:24 := (BUS EXT)1:24
```

I. 2.3.4.2. CHAMP 2. Entrées.

```
ENO
EA      (A)i:j := (e)i:j
EB      (B)i:j := (e)i:j
ERT     (RT)i:j := (e)i:j
ERI     (Ri)i:j := (e)i:j
ECM     (CM)i:j := (e)i:j
ECP     (CP)i:j := (e)i:j
ER1     (R1)i:j := (e)i:j
ER2     (R2)i:j := (e)i:j
EX      (X) 1:24 := (e)1:24
EY      (Y) 1:24 := (e)1:24
EDM     (DM)i:j := (e)i:j
ERA     (RA)i:j := (e)i:j
EAC   [ IF i ≤ 8, (RAP)1:8 := (e )1:8
      [ IF i+j ≥ 17 ∧ (CC)1:8 = 0 ∧ (e)17:8 ≠ 0,
      [       [ (CC)1:8 := (e)17:8
      [       [ bcc := True
      [       [ MCC := (MF)21 = 1

ER8   [ l := (MF)11:3
      [ IF (MF)21 = 1,
      [       (RA)l := R
      [       ELSE,
      [       (R8)l := (s)1

VS    [ IF (p)1:8 ≥ 240 ∨ (p)1:8 ≤ 3,
      [       [ IF (p)1:8 < (ap)1:8 ∨ (ap)1:8 = 0,
      [       [       [ (ap)1:8 := (p)1:8
      [       [       [ (RB)p := 1
      [       [       [ IF ‾MPI‾ ∧ (ap)1:8 ≥ 240
      [       [       [         (apr)1:8 := 0
      [       [       [         ELSE,
      [       [       [         (apr)1:8 := (ap)1:8
      [     ELSE,
      [       | IF(RAP)8 = 0, (BUS EXT)1:24 := (e)1:24
      [       | CALL PERIPH (p)
```

I.2.3.4.3. CHAMP 3. Varia.

VNO	
IAMP	BAMP := True
1AMR	BAMR := True
1MPI	BMPI := True
OAMP	BAMP := False
OAMR	BAMR := False
OMPI	BMPI := False
I	
EX	(X)1:24 := (e)1:24

I. 2.3.4.4. CHAMP 4. Tests.

1R8S	IF(R8)1 = 1 $\vee$ inv $\wedge$ (MF)17:4 $\neq$ CLR8, bs := True
2R8S	IF(R8)2 = 1 $\vee$ inv $\wedge$ (MF)17:4 $\neq$ CLR8, bs := True
3R8S	IF(R8)3 = 1 $\vee$ inv $\wedge$ (MF)17:4 $\neq$ CLR8, bs := True
4R8S	IF(R8)4 = 1 $\vee$ inv $\wedge$ (MF)17:4 $\neq$ CLR8, bs := True
SR8S	IF(R8)5 = 1 $\vee$ inv, bs := True
5A8S	IF(R8)6 = 1 $\vee$ inv, bs := True
7R8S	IF(R8)7 : 1 $\vee$ inv, bs := True
8R8S	IF(R8)8 = 1 $\vee$ inv, bs := True
1S	IF inv, bs := True
1XS	IF (X)1 = 1 $\vee$ inv, bs := True
ELS	IF(équation logique ((X)1, (Y)1, (Z)1, D)) $\vee$ inv, bs := True
DS	IF D $\vee$ inv, bs := True
1XR	IF (X)1 = 1 $\vee$ inv, R := 1
DR	IF D $\vee$ inv, R := 1
TZOS	IF (Z)1:24 = 0 $\vee$ inv, bs := True
DPK	IF D $\vee$ inv, pk := True

I. 2.3.4.5. CHAMP 5. Ordres.

```
ONO

K       IF bmicro,
          (CC) 1:8 := 0
          pk := True
        ELSE,
          k := True

SEQ     séq := True

SMF     smf := True

TZ

1R      R : = 1

0R      R : = 0

LL

LE      (ad1)1:16 := (ad)1:16
        IF (ad)1:12 = 000 ∧ BAMP, (ad1)8 : = 1
        (MP(ad1)1:16)1:24 := (e)1:24

RA+1    (RA)9:16 := x +1

I

DMPI    IF BMPI,  MPI := True
                  (apr)1:8 := (ap)1:8

F       IF(apr)1:8 ≠ 0, fig := True

PK      pk := True

RP      IF (RB)p ≠ 0 ∧ (p)1:8 ≠0,
            (RB)p := 0
            IF (p)1:8 = (apr)1:8,
                (ap)1:8 := minimum{ m / (RB)m = 0 ∧ m < 256}
                IF MPI ∨ (ap)1:8 ≤ 240,
                  (apr)1:8 := (ap)1:8
                ELSE,
                  (apr)1:8 := 0

CLR8    (R8)1:4 := 0
```

I. 2.3.4.6. CHAMP 6. Transfert.

7777
i := 1
J := 24
(e)i:j := (s)1:24

1460
i := 1
j := 8
(e)i:j := (s)9:8

0360
i := 1
J := 8
(e)i:j := (s) 17:8

6003
i := 17
J := 24
(e)i:j := (s)1:8

1403
i := 17
J := 24
(e)i:j := (s)9:8

0303
i := 17
J := 24
(e)i:j := (s)17:8

0101
i := 21
J := 24
(e)i:j := (s)21:4

0201
i := 21
J : 24
(e)i:j := (s)17:4

7007
i := 13
j := 24
(e)i:j := (s)1:12

0707
i := 13
j := 24
(e)i:j := (s) 12:24

1717
i := 9
J := 24
(e)i:j := (s)9:16

3776
i := 1
J := 20
(e)i:j := (s)5:20

7637
i := 5
J := 24
(e)i:j := (s)1:20

```
0102   ⎡ i := 17
       ⎢ J := 20
       ⎣ (e)i:j := (s)21:4
0314   ⎡ i := 9
       ⎢ J := 16
       ⎣ (e)i:j := (s)17 : 8
6060   ⎡ i := 1
       ⎢ j := 8
       ⎣ (e)i:j := (s)1:8
```

I. 2.3.5. MICROS CABLEES

Ecriture mém.
- (ad1)1:16 := (RA)9:16
- IF(ad1)1:12 = 000 ∧ BAMP, (ad1)8 := 1
- $(MP_{(ad1)1:16})1:24$:= (e)1:24

Ecriture m R
- (ad1)1:11 := (RAMR)1:11
- $(MR_{(ad1)1:11})1:24$:= (RT)1:24

INIT
- (RA)1:24 := (RC)1:24
- (ad)1:16 := (RA)9:16
- (e)1:24 := (RC)1:24
- PERFORM Ecrituremém.
- fig := True
- pK := True

WMR
- (RAMR)1 : 11 := (A)14:11
- PERFORM Ecriture MR
- pK := True

RMR
- (RAMR)1:11 := (A)14:11
- (SMR)1:24 := $(MR_{(RAMR)1:11})$ 1:24
- (RT)1:24 := (SMR)1:24
- pK:= True

IRFI
- (ad)1:16 := (RA)9:16
- PERFORM Lectmém
- (e)1:24 := (SM)1:24 + 1
- PERFORM Ecrituremém
- (ad)1:16 := (e)9:16
- CALL PERIPH (p)
- IF (RAP)8=1, (s)1:24 := (BUS EXT)1:24
- (e)1:24 := (s)1:24
- PERFORM Ecrituremém.
- PERFORM RP
- k := True
- fig := True

```
CCM         (ad)1:16 := (RA)9:16
            PERFORM Lectmém.
            (CM)1:24 := (SM)1:24
            PERFORM RP
            k := True

CCC         (ad)1:16 := (RA)9:16
            PERFORM Lectmém
            (A)1:24 := (SM)1:24
            (CC)1:8 := (SM)17:8
            bcc := True
            k := True

WMRP        (RA)1:24 := (A)1:24
            (ad)1:16 := (RA)9:16
            (A)1:24 := x +1
            PERFORM Lectmém.
            (RT)1:24 := (SM)1:24
            pk := True

NOP         k := True

hardmicro   k := False
            EXECUTE ( (RAMR)6:3, INIT, WMR, RMR, IRFI, CCM, CCC,
                                    WMRP, NOP)
            seq := False

Reset       (RB)1 : 255 := 0
            (DM)1 : 24 := 0
            k := True
            BAMP := False
            BAMR := False
            (CC)1:8 := 0
            fig := False
            seq := False
            bcc := False

Initfin     (RB)8 := 1
            (ap)1:8 := 08
            (apr)1:8 := (ap)1:8
```

I. 2.4. Description du Simulateur.

A partir de la définition formelle de l'EPRON, nous avons écrit un simulateur.
Il comprend actuellement 1.400 instructions FORTRAN et 5 routines (ET, OU, EXCLUSION, DECALAGE, INTERRUPTION) très courtes écrites en langage de Base. Une optimisation peut encore doubler ses performances, cependant cet objectif n est pas premier actuellement vu d'une part le temps déjà remarquable pour un cycle (17 ms) et d'autre part la facilité de lecture que nous voulo ns lui donner.

Ce simulateur doit encore être complété par les simulateurs de chacun des périphériques. Pour l'instant nous avons supposé tous nos périphériques banalisés. Une paramétrisation est à l'étude.

I. 2.4.1. Description et fonctionnement.

Il est composé fondamentalement de deux routines distinctes communiquant entr'elles par les variables hardware simulées.

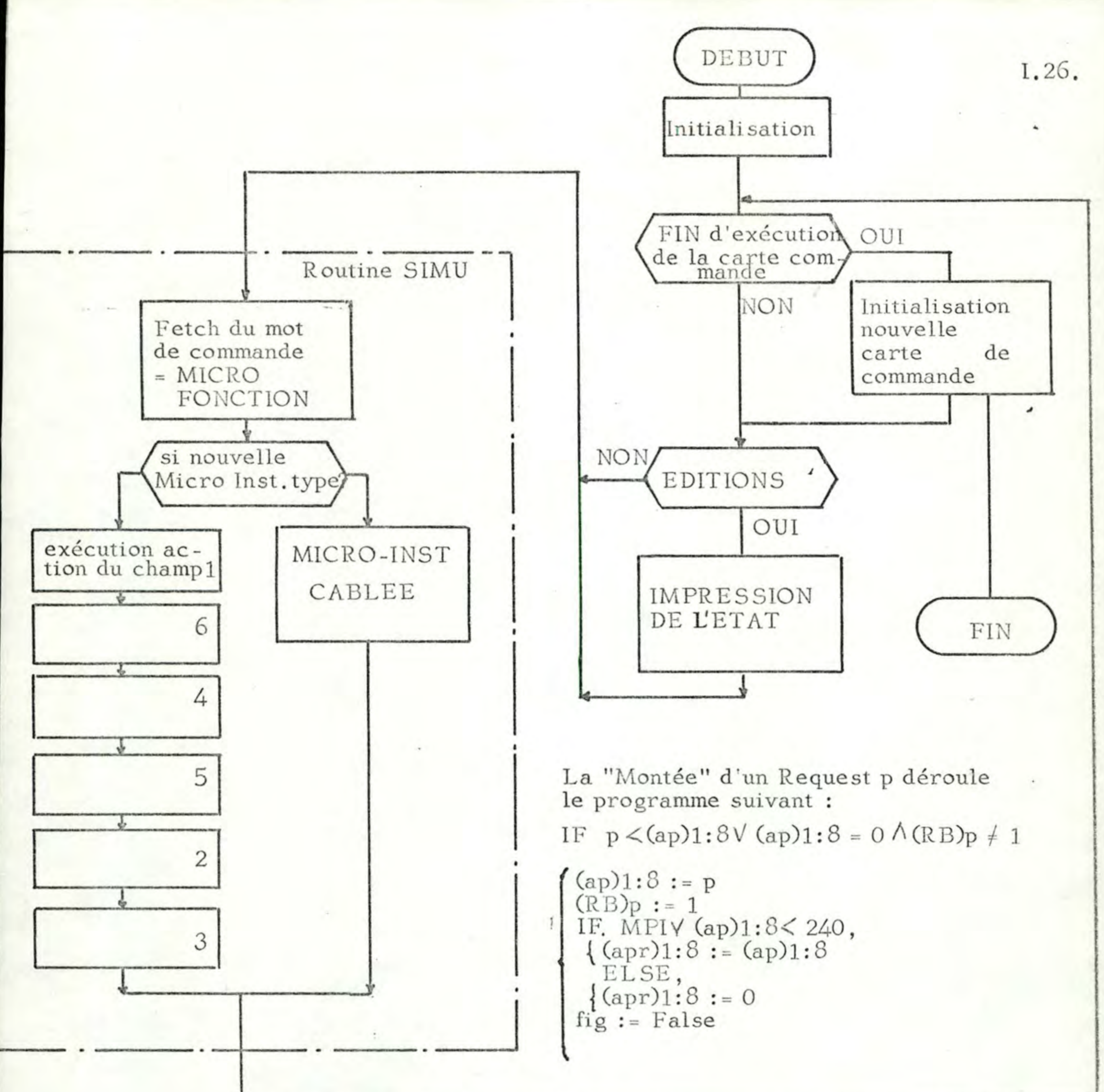
a. La routine SIMU réalise l'exécution d'un sous-temps (= temps élémentaire) en faisant passer les variables (registres, bascules - mémoires etc...) de l'EPRON d'un état à l'état successeur défini par les mécanismes de l'EPRON. Cette routine n'a accès qu'aux variables propres de l'EPRON codées comme elles le seront sur la machine (1).
Cette exécution d'un sous-temps demande une moyenne de 17 m sec. (pour un temps simulé de 250 nsec. - rapport $\simeq$ 70.000 -).

b. Le programme de commande permet de contrôler la simulation en

- donnant accès aux commutateurs simulés du panneau de maintenance
- simulant les interventions des périphériques.
- précisant les éditions demandées.
- permettant d'assembler et de charger des micro et mini-programmes.

---

(1) Cette précision est importante. Nous avons réalisé un simulateur de l'EPRON et non un interprêteur de ses différents langages.



SIMULATEUR ORGANIGRAMME GENERAL

FORMAT DES CARTES DE COMMANDE DU SIMULATEUR

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

TITL — Titre & N.

REST

IED — 0/1

COM — X

INI — 00000000 (DM) 00000000 (A) 00000000 (B) 00000000 (RC) 000000000 (CM) 00000000 (CP) 00000000 (RA)

LMR — M/A — XXXX ßSSSS Ø0000 (1) ßSSSS Ø0000 (2) ßSSSS Ø0000 (3) ßSSSS Ø0000 (4) ßSSSS Ø0000 (5) ßSSSS Ø0000 (6)

LMP — ß/A — XXXX N/S/ß 000000000 SSSS (1) SSSS (2) SSSS (3) XXXX (1) XXXX (2) XXXX (3)

CMR — 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

CMP — 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

adresse — 0 — 1 — 2 — 3 — 4 — 5 — 6 — 7

GO — XXXX

RUN

JALO — XXXX

PAS — XXXX

DUMP — MP/MR — P/C — 0000 0000

REQ — ***

WAIT

FIN

S = Symbole quelconque
X = Nombre decimal
0 = Digit Octal

I.2.4.2. Langage de Commande.

Un langage de commande défini au fur et à mesure des besoins nous permet d'assurer les principales fonctions très souplement. Il est basé sur une indépendance complète des diverses commandes. Celles-ci sont reprises brièvement ci-dessous. A chacune correspond une carte, les 4 premières colonnes indiquant le type de commande, les autres étant réservées à des paramètres ou commentaires.

TITL Titre sur 72 caractères α.N.

Cette carte initialise le N° de Page à 1 et définit le titre indiqué en haut de toutes les pages de la simulation jusqu'à rencontre d'une nouvelle carte TITL.

REST

Cette carte reset les éléments de l'EPRON. Elle simule le fait de pousser sur la touche RESET.

IED 0 ou 1

Cette carte positionne le commutateur IEDM à 1 ou 0.
Ce commutateur mis à 1 inhibe l'entrée de DM et facilite ainsi certains tests.

COM chiffre compris entre 1 et 5

Cette carte positionne le commutateur de mode de fonctionnement de l'EPRON

= 1 progresse d'un sous-temps si on pousse sur RUN
= 2 progresse d'un Temps si on pousse sur RUN
= 3 progresse d'une Ligne Micro si on pousse sur RUN
= 4 progresse d'une Instructions si on pousse sur RUN
= 5 progresse jusqu'à rencontre d'un Figeage.

INI contenu en octal de DM, A, B, RC, CM, CP, RA

Cette carte initialise 7 registres.
C'est la seule façon de charger RC et une facilité pour l'initialisation de certains micro-programmes.

LMR type d'adresse, adresse, 6 fois (mode de représentation, représentation.)

Cette carte permet de charger la Mémoire Rapide (1 mot de 24 bits). Les différents paramètres précisent l'adresse et le contenu,

a. adresse

type d'adresse = A la zone adresse indique alors l'adresse réelle d'implantation.

= M la zone adresse indique le numéro de la Micro Instruction. L'adresse réelle d'implantation est égale à ce numéro multiplié par 8 + 1 (1er sous-temps)

= b̸ la zone adresse n'est pas prise en compte et l'adresse d'implantation est égale à la dernière adresse calculée par une carte LMR augmentée de 1.

Note:lorsque l'adresse réelle (calculée ou non)d'implantation est un multiple de 8, on lui soustrait 8. Cfr. prise en charge des Micro-Fonctions dans l'EPRON.

b. Contenu.

Ce contenu est logiquement divisé en 6 champs de 4 bits . Le contenu de chacun de ces champs est écrit soit en octal soit sous forme symbolique.

mode de représentation = Ø valeur donnée en octal

= b̸ valeur calculée en consultant la table des actions.

LMP type d'adresse, adresse, mode de représentation, représentation.

Cette carte permet de charger la Mémoire Principale (1 mot de 24 bits) Les paramètres précisent l'adresse et le contenu.

a. adresse

type d'adresse = A la zone adresse indique l'adresse réelle d'implantation.

= b̸ la zone adresse n'est pas prise en compte et l'adresse d'implantation est égale à la dernière adresse calculée par une carte LMP augmentée de 1.

b. contenu

mode de représentation

= Ø Le contenu est défini par les 8 digits octaux.

= S Le contenu logiquement divisé en 3 est calculé en consultant la table des Micro-Instructions pour chacune des 3 zones. (1)

= N Le contenu logiquement divisé en 3 est calculé en codant les 3 nombres chacun sur 8 bits.

CMR adresse initiale, 8 fois (valeur octale à charger)

Cette carte permet de charger 8 mots en Mémoire Rapide à partir de l'adresse initiale.

Elle peut être produite par la carte DUMP MR C ......

CMP adresse initiale, 8 fois (valeur octale à charger)

Semblable à CMR mais pour la Mémoire Principale.

---

(1) Cette table est actuellement inexistante, son mode de chargement et le type d'appel restent à préciser.

DUMP nom de Mémoire, type de support, adresse initiale, longueur.

Cette carte permet de vider sur l'imprimante ou sur carte le contenu de zones de Mémoire. Le format de sortie est celui des cartes CMR-CMP. Ceci permet en quelque sorte de garder les micro-programmes miniprogrammes et programmesen code objet absolu.

nom de Mémoire	= MP	pour désigner la Mémoire Principale
	MR	pour désigner la Mémoire Rapide.
type de support	= P	pour désigner l'Imprimante
	= C	pour désigner le perforateur de cartes
adresse initiale	en octal - divisée par 8. Ceci nous donne un alignement standard.	
longueur	en octal - donne le nombre de cartes ou de lignes à sortir.	

REMARQUE.

Le nombre de mots est toujours un multiple de 8 commençant à une adresse multiple de 8.

PAS nombre de pas.

Cette carte permet d'effectuer un "nombre de pas" déterminé avec impression complète de l état de la machine après chaque pas (un pas = un sous-temps élémentaire).

JALO nombre de pas.

Cette carte produit le même effet que la carte PAS mais l'impression est réduite aux éléments permettant de suivre le "cheminement" de la machine.

RUN

Cette carte simule l'appui sur le bouton RUN de l'EPRON. Elle permet donc de faire "tourner" le simulateur jusqu'à ce qu'il rencontre un ordre de Figeage. Il y a alors impression complète de l'état de la machine.

GØ nombre de pas.

Cette carte permet d'effectuer un nombre déterminé de sous-temps avec impression complète de l'état de la machine chaque fois qu'elle se met en figeage.

REMARQUE 1.

Le COM (commutateur de mode de fonctionnement) permet de choisir des impressions après :

chaque sous-temps = 1
chaque Temps = 2
chaque ligne micro = 3
chaque instruction = 4

REMARQUE 2.

La carte PAS n ou les cartes COM 1
GØ n

ont le même effet.

REQ numéro de request.

Cette carte simule la "montée" du request précisé.

WAIT

Cette carte met le simulateur en mode suspendu.
Elle permet de disposer du temps nécessaire pour donner de nouvelles cartes de commandes en fonction des résultats déjà obtenus.

FIN

Sans commentaires.

I.2.4.3. Mécanisme Spécial d'arrêt.

Lorsque le simulateur a pris en compte une carte de commande, il ne peut modifier son traitement que lorsque l'exécution de cette carte est terminée. Il prend alors une nouvelle commande et ainsi de suite. Dans certains cas (bouclages micro-programme avec carte RUN, erreurs donnant lieu à des impressions inutiles) il est précieux de pouvoir interrompre le simulateur en le faisant passer à l exécution de la carte suivante.
Une variable "STOP" est testée avant chaque entrée dans la Routine SIMU. Cette variable mise à 0 lors de la lecture d'une carte de commande peut être mise à 1 (et déclencher l'interruption) manuellement par l'opérateur(dans notre système en mettant l'HORODATEUR en SET durant 3").

I. 3. DESCRIPTION INFORMELLE DE L'EPRON.

Avant propos. .

Chacun sait que pour définir un objet on peut soit le décrire complétement dans sa constitution, soit décrire les propriétés dont il jouit. Cette seconde possibilité est très souvent utilisée en Informatique alors mêmeque l'on croit utiliser la première.

En effet, dans les descriptions à l'intention des programmeurs, nous rencontrons un modèle jouissant de certaines propriétés (dont le programmeur aura à se servir) et non pas l'image du hardware sousjacent. Cette notion de modèle (ou machine virtuelle) se rencontre à tous les niveaux, depuis le schéma logique et la μ-programmation jusqu'aux langages évolués et traitement complexe de fichiers.

La description suivante de notre petit computer est ainsi un modèle à l'intention du μ -programmeur. Le μ -programmeur peut alors construire un nouveau modèle à l'intention du programmeur; par exemple, lui proposer une machine répondant à toutes les caractéristiques (d'un point de vue programmation !) d'une autre déterminée.

I.3.1. Schéma général.

Nous décrivons le computer d'une part par ses composants principaux et d'autre part par les relations entre ces différents composants.
Dans la liste des composants principaux nous trouvons :

- une mémoire principale
- une paire de bus raccordés entre eux par un opérateur de transfert
- une série de registres
- un opérateur arithmétique et logique
- un système de décode μ - instructions et une Mémoire Rapide
- un ensemble de compteurs et clocks
- un système d'interruptions et d'I/O.

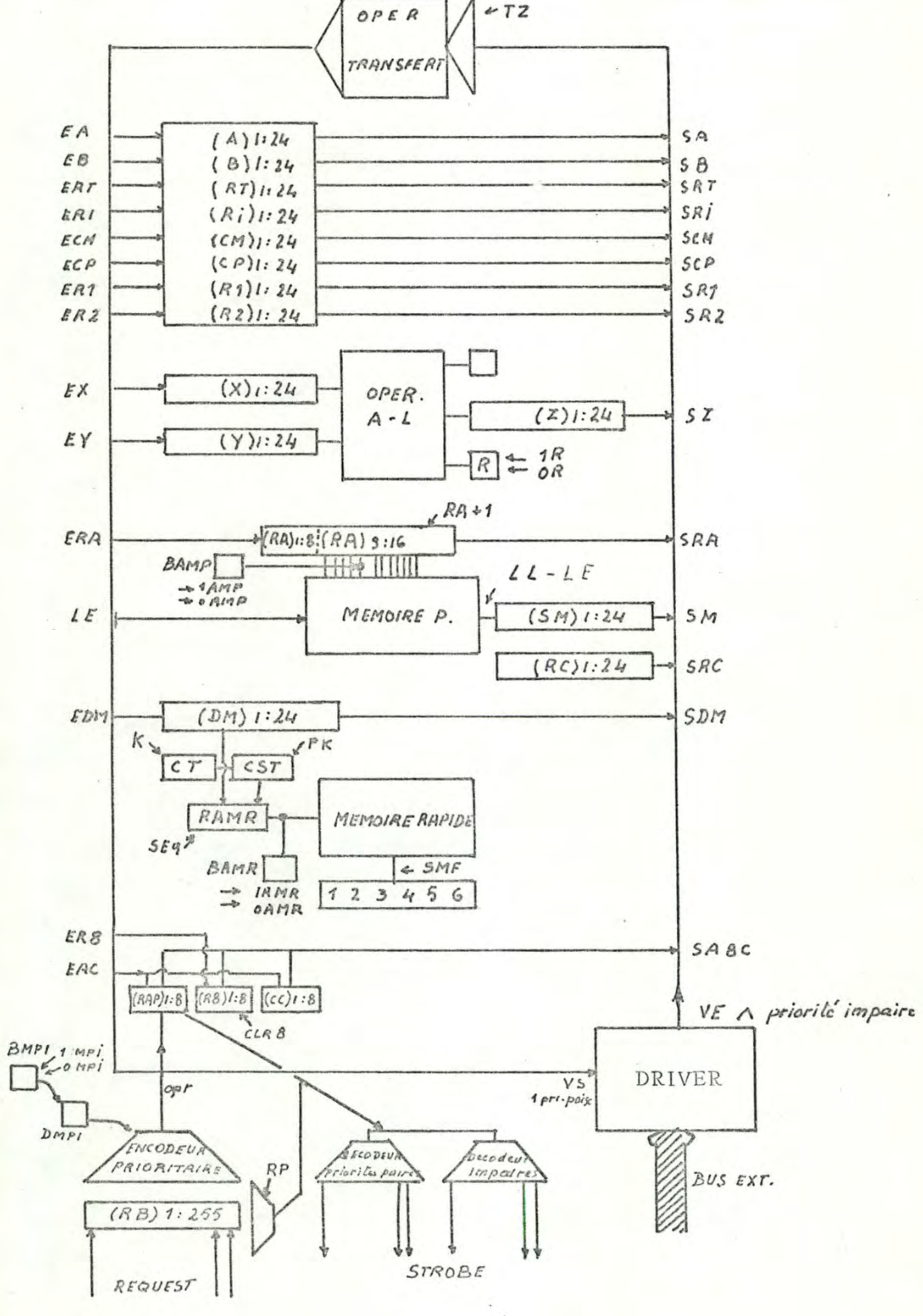
Chacun de ces composants peut être activé par différentes commandes. Nous les divisons en 6 groupes :

1/. Sorties	la sortie d'un registre est raccordée aux bus de sortie
2/. Entrées	l'entrée d'un registre est raccordée aux bus d'entrées
3/. Commandes	de l'opérateur elles positionnent l'opérateur en différents modes.
4/. Tests	un test vérifié devient un ordre.
5/. Ordres	pour lancer certaines opérations ou positionner certainses bascules.

6/. Transfert — définit le mode de passage de l'information entre les bus de sortie et ceux d'entrée.

Exemple :

SRT	le contenu du registre RT est mis sur les bus de sortie.
SA	le contenu du registre A est mis sur les bus de sortie.
ERT	le contenu des bus d'entrée est chargé dans RT en tenant compte du mode de transfert.
EX	le contenu des bus d'entrée est chargé dans X.
ADD	l'opérateur est mis en mode addition des 2 opérandes.
SST	l'opérateur est mis en mode soustraction des 2 opérandes.
DR	si il y a débordement dans l'OAL, alors la retenue est = 1.
1XS	si le 1er bit de X = 1, alors il y a un saut d'un sous-temps.
LL	un lancement lecture est donné à la mémoire.
PK	le compteur des sous-temps est mis à 0 et +1 sur le compteur des temps.
1460	les 8 bits du milieu des bus de sortie sont envoyés sur les 8 bits de gauche des bus d entrée.



OPER
TRANSFERT
TZ
EA
EB
ERT
ERI
ECM
ECP
ER1
ER2
(A)1:24
(B)1:24
(RT)1:24
(Ri)1:24
(CM)1:24
(CP)1:24
(R1)1:24
(R2)1:24
SA
SB
SRT
SRi
SCM
SCP
SR1
SR2
EX
EY
(X)1:24
(Y)1:24
OPER.
A-L
(Z)1:24
SZ
R
1R
0R
RA+1
ERA
(RA)1:8
(RA)9:16
SRA
BAMP
1AMP
0AMP
LL-LE
LE
MEMOIRE P.
(SM)1:24
SM
(RC)1:24
SRC
EDM
(DM)1:24
SDM
K
CT
CST
PK
RAMR
MEMOIRE RAPIDE
SEq
BAMR
SMF
1AMR
0AMR
1 2 3 4 5 6
ER8
EAC
SABC
(RAP)1:8
(R8)1:8
(CC)1:8
CLR 8
VE ∧ priorité impaire
BMPI
1 MPI
0 MPI
DMPI
opr
VS
1 pri. pair
DRIVER
ENCODEUR PRIORITAIRE
RP
Decodeur priorités paires
Decodeur impaires
BUS EXT.
(RB) 1:255
REQUEST
STROBE

I. 3.2. Composants détaillés.

I.3.2.1. Mémoire Principale.

- Mots de 24 bits (+ les parités)
- Adressée par un registre RA
- Accessible à la sortie sous le nom SM
- Activée par deux ordres LL, LE

LL : lancement lecture à l'adresse définie par RA *
l'information se trouve sur SM.

LE : lancement écriture à l'adresse définie par RA*
et contenu donné par les Bus d'Entrée.

RA* est le registre adresse mémoire qui sélectionne effectivement l'adresse mémoire. Sa valeur est égale à celle des 16 bits de poids faible de RA au sous-temps précédent pour autant que le bit BAMP = 0 (cfr. remarque page suivante).

Synchronisation et simultanéité des commandes.

1. entre RA et LL, LE

Puisque l'adresse effective est fonction de la valeur de RA au s-temps précédent, toute ambiguïté est levée.
Exemple :

SR1	ERA	
SR2	ERA	LL
SM		LE

la lecture se fait à l'adresse définie par le contenu de R1,
l'écriture se fait à l'adresse définie par le contenu de R2.

2. entre SM et LL.

Si ces deux commandes sont données dans le même sous-temps, la valeur obtenue à la sortie de SM est celle donnée par le LL.
Exemple :

SR1	ERA	
SR2	ERA	LL
SM	EX	
SM	EY	LL
SZ		LE

le contenu de la Mémoire adressée par R1 est chargé dans X
le contenu de la Mémoire adressée par R2 est chargé dans Y
le résultat Z est écrit en Mémoire à l'adresse R2.

Remarque :

le bit BAMP (bit adresse mémoire principale) permet de modifier l'adresse effective si :

a. - l'adresse donnée dans RA est $\leq$ 15 et

b. - ce bit BAMP = 1

l'adresse effective est alors augmentée de 256.

$(RA*)1:16 := (RA)9:7, (RA)16 \vee \big((RA)9:12 = 0 \wedge BAMP = 1\big), (RA)17:8$

(:= temporisé d'un sous-temps.)

Ce mécanisme permet sur notre machine les modes superviseur-utilisateur. En effet :

a. si BAMP = 0 (superviseur) toute la mémoire principale est accessible.

si BAMP = 1 (utilisateur) l'accès des cellules de mémoire i $0 \leq i \leq 15$ est impossible et devient un accès aux cellules $256 + i$

b. le μ-programme fait correspondre un mot mémoire à chaque code Instruction du langage de Base implémenté. Ce mot mémoire est un point d'entrée du micro-programme d'exécution de cette instruction. L'adresse de ce mot mémoire est égale au code Instruction.

a. et b. $\Longrightarrow$, le point d'entrée dans le μ-programme, pour un même code instruction CI $0 \leq CI \leq 15$ est fonction de BAMP. Le μ-programme peut alors considérer certains points d'entrée comme invalides et générer une interruption de programme.

I.3.2.2. Bus et opérateur de Transfert.

Le premier jeu de Bus (BUS S) est raccordé (moyennant ouverture des portes) à la sortie des différents registres.
Le second jeu de Bus (BUS E) est raccordé (même restriction) à l'entrée des différents registres.
Les 24 bits sont divisés en 6 sections de 4 bits.

Deux digits octaux (= 6 digits binaires) précisent les sections concernées. Une section concernée a son digit binaire mis à 1 sinon il est à 0.

24 bits	1	2	3	4	5	6	6 sections
	1 – 4	5 – 8	9 – 12	13 – 16	17 – 20	21 – 24	
	1er digit octal			2e digit octal			

le code 16 ferait référence aux bits 9 à 20 car :

$(16)_8 = (001110)_2$
123456

et les sections considérées ont les numéro 3,4 et 5 et portent sur les bits 9 à 12, 13 à 16 et 17 à 20.

Le mode de transfert est défini par 4 digits octaux, les deux premiers donnant les sections concernées des bus S et les deux derniers celles des Bus E.

Par. ex. :

le mode 1403 envoie les bits 9 à 16 sur les bits 17 à 24.

L'opérateur de transfert donne les 16 possibilités suivantes :

code symbolique	S	E	
7777			1:24 ⟶ 1:24
1460			9:8 ⟶ 1:8
0360			17:8 ⟶ 1:8
1403			9:8 ⟶ 17:8
0303			17:8 ⟶ 17:8
0101			21:4 ⟶ 21:4
0201			17:4 ⟶ 21:4
7007			1:12 ⟶ 13:12
0707			13:12 ⟶ 13:12
1717			9:16 ⟶ 9:16
3776			5:20 ⟶ 1:20
7637			1:20 ⟶ 5:20
0102			21:4 ⟶ 17:4
0314			17:8 ⟶ 9:8
6060			1:8 ⟶ 1:8

REMARQUE.

Les sections des BUS E non concernées (= non hachurées) ont la valeur 0.

I.3.2.3. Registres.

I.3.2.3.1. Nous avons 8 registres de 24 bits : A, B, RT, CM, CP, RI, R1, R2.
Tous sont accessibles en sortie de la même manière : SA, SB, SRT, SCM, SCP, SRI, SR1, SR2.

Du point de vue hardware, ces registres font partie d'une Scratch Pad. Plutôt que de les définir par leur numéro, nous avons préféré quelques noms rappelant leur usage en micro-programmation.

A, B	pour les accumulateurs principaux et secondaires
RT	Registre Travail
RI	Registre Instruction
CM, CP	Compteur Micro-Programme, Compteur Programme.

En entrée, les 8 registres sont chargés en fonction du code de transfert; c.à.d. que ne sont ouvertes que les entrées correspondant à une valeur non systématiquement nulle (section concernée). (1)

EA, EB, ERT, ECM, ECP, ERI, ER1, ER2.

I.3.2.3.2. Les registres RA (Adresse mémoire) et DM (Décode Micro-cfr. I.3.2.5) peuvent également être lus ou chargés comme les 8 autres registres.

I.3.2.3.3. Un registre de 8 bits R8 peut être chargé et testé 1 bit à la fois; de même ses 4 bits de poids fort peuvent être mis à 0 par l'ordre CLR8.
Pour le chargement, le mode de transfert spécifie :

- le bit à charger (déterminé par les 3 bits de poids faible du champ mode de transfert)
- l'endroit où l'on prend la valeur à charger: bascule R ou bit 1 des BUS S (déterminé par le bit de poids fort du champ mode de transfert).

I.3.2.3.4. Un faux registre de 24 commutateurs du panneau avant, RC, peut être lu sur les BUS S.

```
Ex. :             A  x x x x x x x x, 0 1 0 1 0 1 0 1, 1 1 1 1 0 0 0 0
                  B  x x x x x x x x, 1 0 1 0 1 0 1 0, 1 1 0 0 1 1 0 0

SA EB       1403    donne
tandis que        B  x x x x x x x x, 1 0 1 0 1 0 1 0, 0 1 0 1 0 1 0 1
SA EB       TZ      1403    donnerait
                  B  0 0 0 0 0 0 0 0, 0 0 0 0 0 0 0 0, 0 1 0 1 0 1 0 1
```

---

(1) Cependant l'ordre TZ (Tout à Zéro) permet de supprimer cette sélection des sections en entrée et donc de mettre à Zéro les sections non concernées.

I.3.2.4. Opérateur arithmétique et logique.

Il travaille sur deux opérandes contenus dans les registres (24 bits) X et Y, chargés par EX - EY. Le résultat se trouve en Z et peut être sorti au s-temps suivant le chargement de X et Y par SZ.

Une bascule de retenue peut être mise à 1 ou 0 par les ordres 1R, OR, ou à 1 par les tests.

4 bits de commande positionnent l'opérateur dans un mode d'opération précisé par ces 4 bits. Les modifications de ces 4 bits ont lieu pendant le s-temps 1 et le s-temps 5.

Le débordement (i. e. le bit 0 de Z) peut être testé et donner lieu
soit à un saut de sous-temps DS
soit à une retenue R DR
soit à un PK DPK

De même, une équation logique (1) fonction des bits :

1 de X, 1 de Y, 1 de Z, et D peut être testée et donner lieu à un saut de sous-temps. ELS.

Les principales opérations réalisables sont :

ADD	(Z)	=	(X) +	(Y) + (R)		Addition
ADD1	(Z)	=	(X) +	(R)		+ R
SST	(Z)	=	(X) +	$(\overline{Y})$ + (R)		Soustraction
SST1	(Z)	=	(X) +	$(FFFFFF)_{16}$ + (R)		- $\overline{R}$
SHFT	(Z)	=	(X) +	(X) + (R)		Shift à gauche
NOT	(Z)	=	$(\overline{X})$			inversion
NOR	(Z)	=	$\overline{(X) \vee (Y)}$			
OR	(Z)	=	(X) V	(Y)		
AND	(Z)	=	(X) $\wedge$	(Y)		
EX	(Z)	=	(X) $\veebar$	(Y)		Ou exclusif
ZERO	(Z)	=	0			

---

(1) L'expression de cette équation logique étant encore à confirmer est pour l'instant EL ::= (Z)1 $\veebar$ D.

I.3.2.5. Système de Décode des micro-instructions.

Un registre de 24 bits DM est analysé en 3 Temps T1, T2, T3. Le temps T0 correspond à la prise en charge d'une nouvelle ligne micro dans DM. Chacun des temps T1, T2, T3 analyse une micro-instruction (i.e. une tranche de 8 bits).

Ces 8 bits déterminent une séquence de 8 micro-fonctions écrites en Mémoire Rapide MR (1). Ces 8 micro-fonctions sont analysées à leur tour au cours de 8 sous-temps élémentaires.

Une micro-fonction est un groupe de 24 bits définissant les actions d'un sous-temps.

Ces 24 bits se divisent en 6 champs de chacun 4 bits.

1. Définit le registre qui sortira sur les BUS S-15 possibles.
2. Définit le registre qui est chargé à partir des BUS E - 15 possibles.
3. Définit : aux s.t. 1 et 5 le mode d'opération de l'opérateur; aux autres s.temps un second registre d'entrée ou un ordre (champ Ramasse-tout).
4. Définit les différents tests.
5. Définit les ordres.
6. Définit le mode de transfert.

I.3.2.6. Compteurs et Clocks.

1. sous-temps.
2. temps.
3. compteur de cycle.

Le compteur de sous-temps est un compteur modulo 9.
Nous avons donc 9 sous-temps. Le sous-temps 0 est inutilisable par micro-programme.
Les sous-temps 1 à 8 balayent successivement la séquence de micro-fonctions en MR.

Le compteur de Temps est un compteur modulo 4.
Il définit les 3 micro-instructions d'une ligne.
Le T0 est utilisé par cablage pour la prise en charge d'une nouvelle ligne micro - (en séquence ou en interruption).

Le décompteur compteur de cycle est à 8 bits (0-255).
Il peut être chargé par EAC mais n'est validé que par le 1er ST0 ou T0 qui suit EAC. (2).

Le bit de poids fort du mode de transfert détermine le mode de bouclage :

= 0 mode micro-instruction.

= 1 mode ligne micro-instruction.

---

(1) La MR est une mémoire de 2 K mots de 24 bits.
Elle est donc adressée par 11 bits, 8 (poids fort) venant de DM - définissant la micro-instruction en cours - et 3 (poids faible) provenant du compteur de sous-temps (st 1 à st8).

(2) EAC vient de Entrée AC - Le pseudo registre AC est en fait la concaténation des deux registres RAP (cfr. système d'interruption et d'I/O) et CC. RAP cadré à gauche et CC cadré à droite. Le mode de transfert permet de choisir le registre voulu en déterminant les sections concernées des BUS E.

En plus de EAC, 4 ordres permettent d'agir sur ces différents compteurs S, PK, SEQ, K.

La table ci-dessous donne une première idée de l'action de ces différents ordres sur les compteurs. Pour une définition précise voir la définition formelle.

			CC	Sous-temps	Temps
CC = 0		S	-	+ 1	-
		PK	-	0	+ 1
	mode micro	SEQ	-	0	-
	PK ∧ T3 ∨	K	-	0	0
		EAC	chargé	-	-
		S	-	+ 1	-
	mode ligne	PK	-	0	+ 1
	micro	SEQ	-	0	-
	PK ∧ T3 ∨	K	-	0	0
		EAC	chargé	-	-
CC ≥ 1		S	-	+ 1	-
		PK	- 1	0	-
	mode micro	SEQ	- 1	0	-
		K	0	0	+ 1
		EAC	-	-	-
		S	-	+ 1	-
	mode ligne	PK	-	0	+ 1
	micro	SEQ	-	0	-
	PK ∧ T3' ∨	K	- 1	0	0
		EAC	-	-	-

Remarque :

- pour boucler sur une ligne micro, il faut charger CC dans la ligne micro précédente.
- pour boucler sur une micro, il faut charger CC dans la micro précédente.
- les valeurs 0 et 1 sont équivalentes dans CC i.e. boucler 0 fois = passer 1 fois comme s'il n'y avait pas de compteur de bouclage.

  Si le CC est chargé à 20 on exécutera 20 fois la micro.

I.3.2.7. Système d'interruption et d'I/O.

Cfr. schéma général.

I. 3.2.7.1. Description :

RB	Request Bank		256 bits
APR	adresse périphérique Request		8 bits
BUS EXT	Bus extérieur		24 lignes
RAP	registre Adresse Périphérique		8 bits
256	Strobes	(fil)	
256	Requests	(fil)	
256	ResetsPriorité	(fil)	
RP	Reset Priorité	(ordre)	
VE	Validation Entrée	Commande champ 1	
VS	Validation Sortie	Commande champ 2	
MPI	Masque Priorités inférieures		1 bit
BMPI	Validation du Démasquage des Priorités inférieures		1 bit

Les commandes possibles sur ces organes sont :

EAC	Entrée de RAP qui charge les 8 bits de RAP à partir des 8 bits de gauche des BUS E (cfr. remarque 2 section I.3.2.6).
RP	Reset la priorité dont l'adresse est donnée dans RAP.
VE	Envoie un strobe de ≠ = à celui de RAPet ouvre la porte de communication des Bus Extérieurs vers les Bus S.
VS	Envoie un strobe de ≠ = à celui de RAP et ouvre la porte de communication des BUS Evers les Bus Extérieurs.
OMPI 1MPI	Met à 0 (à1) la Bascule BMPI
DMPI	Met à 1 le Masque des Priorités inférieures.
MPI	Met à 0 le Masque des Priorités inférieures.

I.3.2.7.2. Fonctionnement.

Les requests sont donc mémorisés et éventuellement masqués. Si un ou plusieurs requests peuvent apparaître à travers le masque, un encodage prioritaire a lieu et donne dans APR l'adresse du request le plus prioritaire.

APR est utilisé en TO (y voir définition précise) pour donner une adresse de débranchement.

Cette adresse (≈ APR x 4 + constante) est utilisée pour le calcul de la prochaine ligne micro à exécuter (sans modification de CM)

Le système interruption I/O permet donc :

1. de masquer ou non des Requests.
2. d'exécuter une ligne micro dont l'adresse est fonction du Request.
3. d'exciter 1 des 256 strobes.
4. de resetter la mémorisation du Request.
5. de communiquer des Bus E → Bus Externes.
6. de communiquer des Bus Externes → Bus S.

Remarque.

Les mots mémoires associés à un Request sont appelés mots réservés exécutifs.
Les groupes de 3 mots qui suivent un mot réservé exécutif sont appelés mots réservés pour paramètres.

Remarque :

1. Plusieurs Priorités seront habituellement utilisées pour 1 seul périphérique.

   par ex. 1 priorité de commande
   1 priorité de Data Transfert
   1 priorité de Mot d'Etat.

2. En reliant le strobe au Request d'une même priorité on peut créer une priorité dite "priorité interne".

3. Le Strobe envoyé sert à valider (clocker) le chargement d'un registre du périphérique à partir de la valeur des BUS EXT. dans le cas d'un VS ou à valider la sortie d'un registre périphérique sur les BUS EXT mettre sa sortie en basse impédance dans le cas de technologie TRISTATE) si il s'agit d'un VE.

I. 3.3. Commandes.

I.3.3.1. Tests :

A partir de la valeur d'une équation logique le plus souvent réduite à un terme, un décision est prise.

Les équations logiques sont :

bit 1 de R8, bit 2 de R8 .... bit 8 de R8,
Débordement, bit 1 de X, NOR des 24 bits de Z, EL, Faux.

Les décisions portent sur :

saut d'un sous-temps (S),
positionnement à 1 de la bascule report (R)
mise à 0 du compteur de sous-temps (PK).

16 combinaisons sont permises :

IR8S	
2R8S	
3R8S	
4R8S	
5R8S	
6R8S	
7R8S	
8R8S	
1XS	bit 1 de X
1XR	
ELS	équation logique
DS	débordement
DR	
DPK	
TZOS	test Z = 0
1S	1)

S	Saut inhibe l'exécution du s.temps suivant en séquence.	
Un ordre I	Inversion provoque l'inversion de la condition de test.	
D = R	signifie si débordement = 1	alors le Report = 1
D = R	avec I " " " " = 0	alors le Report = 1

N.B. : Si la condition de gauche n est pas réalisée, aucun passage n'a lieu à droite (ni à 1 ni à 0).

---

1) Si l'ordre I n'est pas présent, il n'y a pas de saut.
Si l ordre I est présent, il y a saut d'un sous-temps.
IS est en fait câblé sur une valeur toujours = 0.

I.3.3.2. Ordres.

Ils se répartissent logiquement en 4 groupes suivant l'organe sur lequel ils agissent.

LL		Lancement Lecture
LE	Mémoire	Lancement Ecriture
RA+1		Addition de 1 sur RA
DMPI	I/O	Démasque Priorité Inférieure
RP		Reset Priorité
I		Inversion de la condition de test
SMF		Sortie Micro Fonction
PK	Contrôle	agissent sur CC, Compteur s-temps et tem
SEQ		
K		
F		Figeage
1R		Mise à 1, 0 du Report
OR	Data	
TZ		Transfert Zéro
CLR8		Clear R8

SMF — Inhibe l'exécution du sous-temps suivant.

La Micro fonction n'est pas à exécuter mais est une constante à charger dans RT.

F — Figeage (n'est significatif que si aucune priorité non masquée n'est présente). Cet ordre fige le calculateur après l'exécution du sous-temps contenant F.
Le défigeage se fait soit par un bouton soit par l'arrivée d'une priorité non masquée; il y a alors poursuite en séquence.

I.3.3.3. Entrées et Sorties des registres et bascules.

Sorties 1er champ	Entrées (1) 2e champ	Entrées (2) 3e champ
SA	EA	
SB	EB	1MPI
SRT	ERT	1AMP
SRI	ERI	1AMR
SCM	ECM	OMPI
SCP	ECP	OAMP
SR1	ER1	OAMR
SR2	ER2	I
SZ	EX	EX
SRA	EY	/.
SDM	EDM	.
SRC	ERA	.
SM	ER8	:
SA8C	EAC	.
VE	VS	NO

NB. Aux s.t.1 et 5 le champ 3 est utilisé pour définir le mode d'opération de l opérateur A-L.

Le pseudo registre A8C est la concaténation des 3 registres RAP, R8 et CC; il permet de les lire, la sélection s'opérant par le mode de transfert.

Le pseudo registre AC est le regroupement sur 24 bits des 2 registres RAP (cadré à gauche) et CC (cadré à droite).
Les 8 bits du milieu sont "déconnectés".

VE et VS (Validation Entrée - Validation Sortie) permettent la communication entre les BUS Externes et Internes.

Les bits BMPI, Bit Masque Priorités Inférieures (cfr. I/O).
BAMP Bit Adresse Mémoire Principale (cfr. Mémoire)
BAMR Bit Adresse Mémoire Rapide (cfr. M.R.)

sont mis à 1 ou à 0 par les commandes

1MPI, 1AMP, 1AMR et OMPI, OAMP, OAMR.

I. 3.4. Rappel de Notation pour la micro-programmation.

Nous avons 3 micro-instructions dans un mot mémoire.

Elles sont notées | micro 1 | micro 2 | micro 3 | soit par un symbole mnémonique, soit par un n° correspondant à la définition donnée (au moment considéré) dans la M. Rapide.

Une micro est définie par 8 lignes. Chaque ligne correspond à 1 sous-temps, elle peut éventuellement être vide.

Dans chacune des lignes on ne peut trouver qu'une commande ou ordre de chacun des champs.

Sortie	Entrée 1	(Opérateur / Entrée 2 / Ordre 2)	Test	Ordre	Transfert

Ces commandes ou ordres sont écrites de façon mnémonique ou en référence à l'encodage vu précédemment.

Ex. : Décalage de 1 bit à gauche sur R1, R2.

	Sorties	Entrées	Varia	Tests	Ordres	Transfert
1	SR2	EX	SHFT			7777
2	SZ	ER2		DR		7777
3	SR1	EX				7777
4	SZ	ER1			PK	7777

1 l'opérateur est mis en SHFT $\Longrightarrow$ (Z) = (X) + (X) + (R)
X est chargé de la valeur de R2.

2 R2 est rechargé de son ancienne valeur décalée de 1 bit à gauche, le débordement éventuel positionne le Report et permet ainsi le transfert du bit de gauche de R2 dans le bit de droite de R1.

3et4 R1 subit la même opération paramétrée par le Report.

I.3.5. Définition de TO.

Pour cette définition nous employons un langage proche de celui de notre μ -programmation. Il y a quelques différences que l'on voudra bien remarquer. De plus comme TO est cablé, tout se passe comme si il réalisait cette μ -instruction.
En fait il la réalise autrement.

Nous devons distinguer 3TO : TON, TOI, TOB
correspondant respectivement au cas Normal, Interruption et Bouclage et classés dans l'ordre de priorité croissante.

I.3.5.1. TON — si aucune interruption non masquée n'est présente et si il n'ya pas bouclage sur une ligne micro :

Sorties	Entrées	Varia	Tests	Ordres	Transfert
SCM	ERA			OR	1717
	EX			LL	7777
SM	EDM			RA+1	7777
SRA	ECM			PK	1717

DM est chargé en fonction de (CM) (Compteur Micro) et (CM) est incrémenté de 1 . (X) et (R) sont mis à 0.

I.3.5.2. TOI — si il n'y a pas bouclage sur une ligne micro et si APR≠0

Sorties	Entrées	Varia	Tests	Ordres	Transfert
SAPR	ERAP [1]			OR	spécial (1)
SRAP	ERA				spécial (1)
SM	EDM			LL	7777
	EX			PK	7777

DM et RA sont chargés en fonction de (APR) , (X) et (R) mis à 0
APR (Adresse Priorité Request) donne le numéro du request présent le plus prioritaire. (2).

I.3.5.3. TOB — Bouclage sur une ligne micro.

Sorties	Entrées	Varia	Tests	Ordres	Transfert
	EX			OR	7777

---

1) SAPR, ERAP, ERA] se fait en respectant la position relative des bits dans les différents registres. cfr. figure A. (Pg. suivante)

2) La priorité n'a pas été remise à 0 pendant TOI.

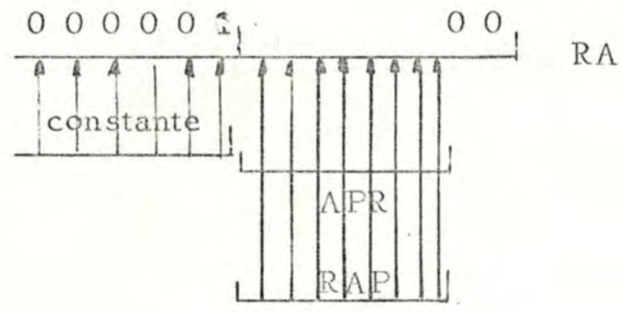


Fig. A

I.3.6. Définition de st0 et st8.

Même avertissement que pour T0.

EY OR

Quand il n'y a eu ni K ni PK dans les 8 st on suppose qu'il y a un PK implicite ou st 8.
Ceci permet d'avoir encore un ordre possible au st8.

Rappel :

La description de TO et STO se fait dans un langage fonctionnel et non descriptif du hardware. La hardware est câblé pour exécuter cette fonctionnalité avec efficience maximum sans nécessairement utiliser les modèles de circuit définis dans ce manuel.

Note :

La principale fonction de st0 est de prendre en charge les 8 bits de poids forts du :

Registre
Adresse
Mémoire
Rapide
(RAMR)

donnés par le code micro de DM (gauche, milieu ou droite suivant T1, T2, T3).

Ce chapitre 1 définissait le support hardware de notre système. Avant d'étudier le dynamisme au niveau micro-langage (section II 4.2.) et mini-langage (chapitre III), les premières sections du chapitre II vont nous familiariser avec les mécanismes de l'EPRON et la technique de micro-programmation.

Des résultats de simulation sont disponibles au secrétariat de l'Institut d'Informatique (1). Ils permettent de confronter les mécanismes de l'EPRON à l'image que nous nous en faisons. Cette confrontation étant superflue dans une première approche, nous n'avons pas insérés les simulations dans ce chapitre II.

Ceux qui voudraient approfondir la connaissance active de l'EPRON peuvent soit écrire quelques micro-programmes et les tester sur ce simulateur, soit suivre pas à pas les exemples déjà simulés.

---

1. Institut d'Informatique F.U.N.D.P.
   8, rempart de la Vierge,
   B - 5000 NAMUR.

CHAPITRE II

REALISATION DE MICRO-PROGRAMMES.

1. ENVIRONNEMENT.
1.1. Format des Instructions.
1.2. Topologie de la Mémoire Centrale.
1.3. Principe recherche nouvelle Instruction.
1.4. Convention d'affectation des registres.
1.5. Adressage.
1.5.1. Indirect.
1.5.2. Indexé.
1.5.3. Direct absolu.
1.5.4. Indexé indirect.

2. INSTRUCTIONS ELEMENTAIRES.
2.1. Load accumulateur.
2.2. Add accumulateur.
2.3. Décrément Mémoire.

3. MICROPROGRAMMES PLUS COMPLEXES.
3.1. Input-Output avec Cadrage et Data Chaining
3.1.1. Organigramme et Micro-Programmation.
3.1.2. Remarques et Commentaires.

3.2. Conversion binaire - BCD.
3.2.1. Organigramme.
3.2.2. Micro-programme.

3.3. Conversion BCD - binaire.
3.3.1. Organigramme.
3.3.2. Micro-programme.

4. PREMIERES CONCLUSIONS.
4.1. Dynamisme au niveau mini-langage.
4.2. Dynamisme au niveau micro-langage.
4.2.1. RNI statistiques.
4.2.2. RNI Trace.
4.2.3. Conditions d'une application optimale.

\- - - - - - - -

CHAPITRE II

REALISATION DE MICRO-PROGRAMMES.

II. 1. ENVIRONNEMENT.

Pour mettre en évidence les mécanismes de l'EPRON, nous développons dans ce chapitre quelques exemples de micro-programmes.
Ces micro-programmes assurent l'interprétation d'une architecture virtuelle, c'est-à-dire d'un ensemble - Décor virtuel - Jeu d instructions.

Le Décor virtuel est l'ensemble des éléments visibles au niveau du programmeur ; le Jeu d'instructions définissant l'ensemble des opérations réalisables sur ces éléments.

L'architecture que nous présentons ici n'est qu'une ébauche sommaire et incomplète. Elle nous servira uniquement de point de repère.

II. 1.1. Format des Instructions.

Nos instructions de longueur fixe, 1 mot de 24 bits, sont divisées en 3 champs.

1 — 6	7, 8, 9 — 12	13 — 24
code opération	contrôle adresse	Adresse ou Paramètre

Les bits 1 à 6 définissent le code opération.

Les bits 7 à 12 définissent le contrôle de l'adresse.

Ils indiquent :

bit 7	indirection si bit 7 = 0
bit 8	indexation si bit 8 = 0
bit 9 à 12	numéro du registre d'index (si bit 8 = 0)
	bits de poids fort de l'adresse (si bit 8 = 1).

N.B. : Nous travaillons en préindexation.

Les bits 13 à 24 définissent l'adresse ou un paramètre.

II. 1.2. Topologie de la Mémoire Centrale.

Physiquement banalisée, la Mémoire Centrale est divisée logiquement en 4 domaines :

1. Points d'entrée des micro-programmes :
   $0 \leq$ adresse ≤ 255.
2. Zone de continuation pour les micro-programmes longs et tables diverses :
   $256 \leq$ adresse ≤ 1023.
3. Points d'entrée des interruptions et paramètres I/O
   $1024 \leq$ adresse ≤ 2047.
4. Software : $2048 \leq$ adresse ≤ 65535.

1. A chaque code opération sont associées 4 mots mémoire consécutifs d'adresse =

code opération x 4 +	0	si bit 7 = 0 et bit 8 = 0
	1	si bit 7 = 0 et bit 8 = 1
	2	si bit 7 = 1 et bit 8 = 0
	3	si bit 7 = 1 et bit 8 = 1

   Ces mots mémoire contiennent la 1ère ligne micro (et souvent l'unique) du micro-programme chargé d'interpréter l'instruction.

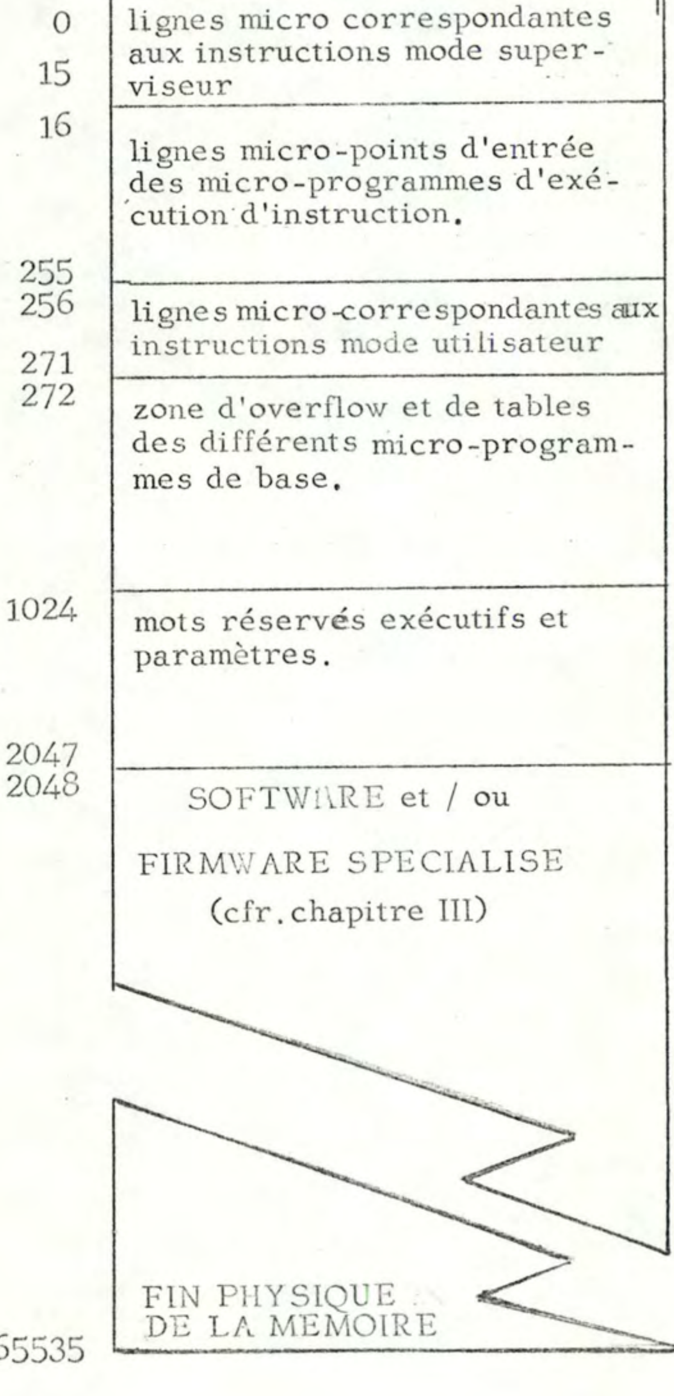
2. Si il faut plus d'une ligne micro pour interpréter l'instruction, le micro-programme se branche dans la zone de continuation.

3. En cas d'interruption n° i prise en compte par l'encodeur prioritaire, il y a exécution de la ligne micro d'adresse = i x 4 + 1024.

   Les mots d'adresse = i x 4 + 1025 à i x 4 + 1027 sont réservés à des paramètres tels que pointeur buffer, pointeur CCW etc....

4. La zone software contient tous les programmes système-utilisateur et est considérée comme zone de données par le micro-programme.

TOPOLOGIE DE LA MEMOIRE CENTRALE



II. 1.3. Principe recherche nouvelle Instruction.

Comme nous l'avons vu dans le § précédent, les 256 premiers mots de la mémoire sont les points d'entrée des micro-programmes de de décode d'instruction.
La recherche d'une nouvelle instruction se déroule en 4 phases :

1. en fonction du compteur de programme (CP) aller lire l'instruction en Mémoire.
2. charger l'instruction dans un registre d'analyse (RI).
3. Brancher le micro-programme à l'adresse = code opération x 4 + (bit 7) x 2 + bit 8.
4. Incrémenter le CP de 1.

Ceci se traduit par la micro suivante :

SCP	ERA	DMPI	1717
SM	ERI	LL	7777
SM	ECM	RA+1	6003
SRA	ECP	K	1717

les 8 bits de poids fort de l'instruction sont chargés dans CM. Ainsi la prochaine ligne micro exécutée sera celle correspondant au pt.d'entrée du micro-programme interprétant cette instruction, sauf dans le cas d'interruptions micro-programme, voir même d'interruptions de programme démasquées par l'ordre DMPI.

II. 1.4. Convention d'affectation des registres.

Etant donné la possibilité d interruption entre 2 lignes micro, nous devons définir les registres dont les valeurs doivent être considérées comme perdues d'une ligne à l'autre. Ces registres servent de zone manoeuvre pour les différents programmes, tant ceux d'exécution d instruction que ceux d'interruption. Ce sont principalement :

X] entrée de l opérateur
Y]
et donc Z sortie de l'opérateur

RT Registre Travail

RA Registre Adresse Mémoire.

(R8)7:2 Ces 2 bits permettent de mémoriser des conditions.

Les autres registres se divisent en 2 classes. L'une visible du programmeur :

(RA)1:4	=	Code Condition
(CP)9:16	=	Compteur Programme
(A)1:24	=	Accumulateur
(B)1:24	=	Extension accumulateur.

L'autre à l'usage exclusif du micro-programmeur :

CM	Compteur Micro-programme
RI	Registre Instruction
R1	Registres
R2	

DM, (R8)5 : 2, CC, RAP etc...

II. 1.5. Adressage.

Trois micro-instructions permettent de réaliser les 4 combinaisons possibles pour une instruction référence mémoire.

- Indexage indirect
- Indirection absolue
- Indexage direct
- Direct absolu.

Toutes trois supposent le champ adresse chargé dans (R1)9 : 16 et donnent l'adresse effective dans (RA)9:16.

II. 1.5.1. Indirection absolue. I

```
SRI   ERA              1717
SM    ERA         LL   1717
                  PK
```

II. 1.5.2. Indexage direct. IDX

```
SRI   EX   ADD         1403
SZ    ERA         SMF  0201   (RA)21:4:=n° du registre
      constante base adresse des registres
SRT   ERA              3776   (RA)9:12:=adresse du reg.
SM    EX   ADD    LL   1717
SRI   EY               0707
SZ    ERA         PK   1717
```

II. 1.5.3. Direct absolu. DA

```
SRI   ERA         PK   1717
```

II. 1.5.4. Indexage indirect IDXI

```
SRI   EX   ADD          1403   ┐
SZ    ERA        SMF    0201   │
  Constante base adresse des registres   │  indexage
SRT   ERA               3776   │
SM    EX   ADD   LL     1717   │
SRI   EY                0707   ┘
SZ    ERA               1717
SM    ERA        LL     1717      indirection
```

Remarque :

Les 16 registres se trouvent en Mémoire Centrale.
Leur adresse est donnée en concaténant une constante à leur numéro.

II. 2. Instructions Elémentaires.

3 Instructions (Load, Add et Incrément Mémoire) sont données ci-dessous comme exemples de micro-programmation.

II. 2.1. Load accumulateur.

Cette instruction charge l'accumulateur du contenu Mémoire.
Le code condition est positionné en fonction de la valeur chargée (V):

bit	1	= 0	
	2	= 1	si $V = 0$
	3	= 1	si $V > 0$
	4	= 1	si $V < 0$

Elle est exécutée par une micro-instruction d'adressage suivie de la micro-instruction LD suivie elle même de RNI.

LD	SM	EX	ADD		LL	7777	
	SM	EA	I	TZOS	CLR8	7777	
	SRC	ER8	I		PK	2R81	$A = 0$
				1XS			
	SRC	ER8			PK	3R81	$A > 0$
	SRC	ER8			PK	4R81	$A < 0$

Ces 3 micro-instructions étant sur une même ligne micro.

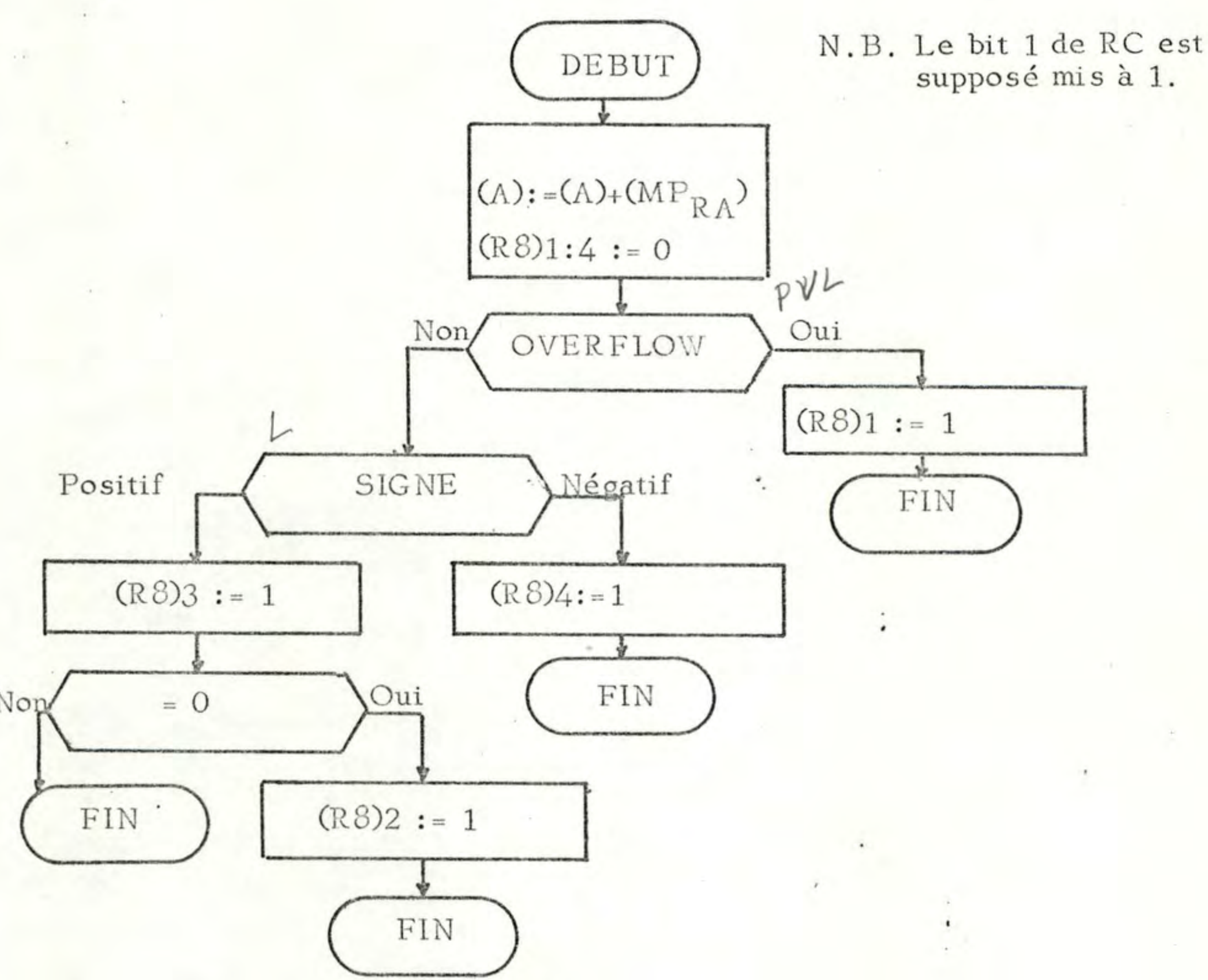
IDXI	LD	RNI	Load avec Indexage et Indirection
I	LD	RNI	Load avec Indirection Absolue
IDX	LD	RNI	Load avec Indexage direct
DAA	LD	RNI	Load absolu direct

II. 2.2. Add Accumulateur.

Cette instruction additionne sur l'accumulateur le contenu Mémoire. Le code condition est positionné en fonction du résultat.

bit		
1	= 1	si overflow
2	= 1	Résultat = 0
3	= 1	si Résultat $\geqslant 0$
4	= 1	si Résultat < 0

Cette instruction est basée sur la micro-instruction ADD dont voici l'organigramme.



ADD	SM	EX	ADD		LL	7777	
	SA	EY			CLR8	7777	
	SZ	EA	I	ELS		7777	
	SRC	ER8			PK	1R81	overflow
			ADD	1XS	I		
	SRC	ER8			PK	4R81	< 0
	SRC	ER8	I	TZOS		3R81	≥ 0
	SRC	ER8			PK	2R81	$= 0$

IDXI	ADD	RNI
I	ADD	RNI
IDX	ADD	RNI
DA	ADD	RNI

Contenu des 4 lignes micro. associées à l'instruction ADD.

II. 2.3. Décrément Mémoire.

Cette instruction décrémente de 1 le mot mémoire défini par le champ adresse. Si le résultat est = 0 il y a saut d'une instruction c'est-à-dire, incrément de 1 du compteur programme.

La micro-instruction DM assure l'exécution principale.

DM	SM	EX	SST1		LL	7777	Lecture mot mémoire
	SZ			TZOS	LE	7777	Ecriture mot décré-menté
					PK		≠ 0
	SCP	ERA				1717	= 0
					RA+1		incrément de CP
	SRA	ECP			PK	1717	

1 ⟶	IDXI	DM	RNI
2 ⟶	1	DM	RNI
3 ⟶	IDX	DM	RNI
4 ⟶	DA	DM	RNI

1 Décrément Mémoire avec Indexage suivi d'indirection.

2 Décrément Mémoire avec Indirection absolue

3 Décrément Mémoire avec Indexage direct.

4 Décrément Mémoire direct absolu.

II. 3. MICROPROGRAMMES PLUS COMPLEXES.

Pour montrer l'utilisation de l'EPRON au mieux de ses possibilités, nous donnons ici 3 applications mises au point et testées sur le simulateur.

Il s'agit du micro-programme d'interruption pour Input-Output avec Data chaining et de 2 micro-programmes de conversion binaire -BCD et BCD-binaire.

Un compromis a été réalisé entre une microprogrammation simple (avec risques de non efficience) et l'usage de mécanismes plus efficients (et moins lisibles).

Les résultats de simulation disponibles aux FNDP permettent de suivre les différentes étapes de chacun des micro-programmes et ce pour plusieurs paramètres.

II. 3.1. Input-Output avec cadrage et Data Chaining.

mot réservé exécutif →

50	51. 52 53 - 54 - 55 56 - 57 - 58
CCW COUNT	CCW POINTER

p	L1 *	1° CCW
	n1	
p	L2 *	2° CCW
	n2	
p	L3 *	3° CCW
	n3	
p	L4 *	4° CCW
	n4	

p numéro de la priorité à envoyer quand tout est terminé.

Li * complément à 2^{15} de la longueur.

ni adresse de la zone.

Lorsque le Request est pris en compte, la ligne micro correspondante est exécutée. Cette ligne micro définit le type d'opération I/O à effectuer:

Code	Type	Description
50 51 53	Input	caract. de gauche dans le mot.
50 51 54	Input	caract. du milieu dans le mot.
50 51 55	Input	caract. de droite dans le mot.
50 51 56	Output	caract. de gauche dans le mot.
50 51 57	Output	caract. du milieu dans le mot
50 51 58	Output	caract. de droite dans le mot.

Si le CCW COUNT est ≥ 128, elle envoie une priorité n° p et termine. Sinon, elle incrémente L* (complément de la longueur) de 1 et, si L* ≥ 2^{15}, met à jour le CCW count (+ 1) et le CCW pointer (+ 2).

En fonction du code micro, elle réalise une des 6 fonctions et écrit dans le mot réservé exécutif la ligne micro qui sera exécutée lors du prochain Request (54 succède à 53, 55 à 54 et 53 à 55, de même 57 à 56, 58 à 57 et 56 à 58).

L'organigramme donne les précisions voulues.

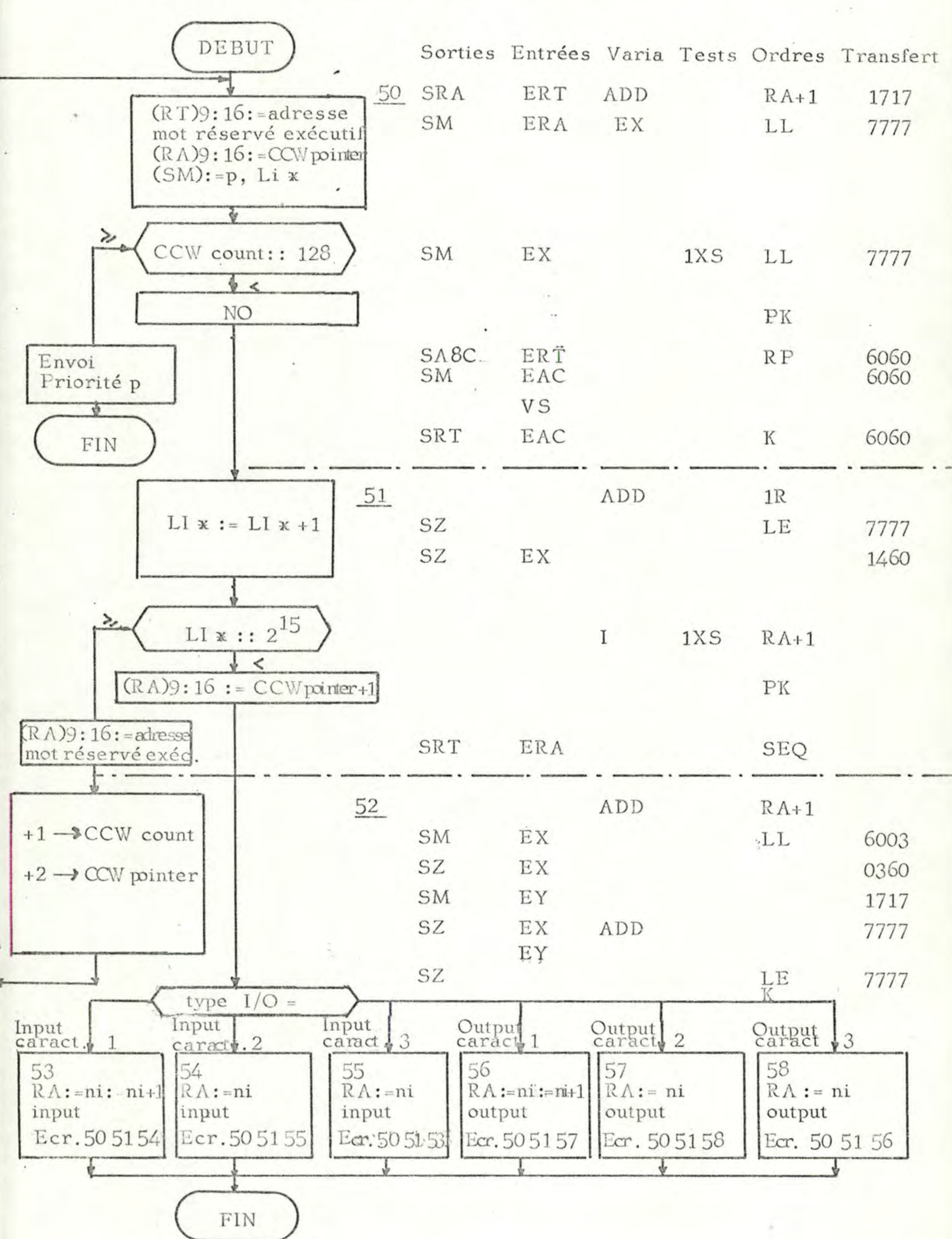
Le temps de prise en compte d'un Request est de 11,75 μ sec. sans Data Chaining et de 22 μ secondes avec Data Chaining.

Input — 1er caractère (gauche)

```
53                      ADD       1R
       SM    EX                   LL    7777
       SZ    ERA                  LE    7777   RA := ni := ni + 1
       VE                         LE    0360   Input caractère
       SRT   ERA                  SMF   1717
             50        51         54           prochaine ligne micro
       SRT                        LE    7777   mise à jour
```

2e et 3e caractère (milieu, droite)

```
54     SM    ERA                  LL    1717   RA := ni
- 55   SM    EDM                  LL    7777   lecture du mot
                                          14 ← 54
       VE    EDM                        03 03 ← 55   insertion du caractère
       SDM                        LE    7777   écriture du mot
       SRT   ERA                  SMF   1717
                                  55          ← 54
          50          51          53          ← 55   prochaine ligne micro
       SRT                        LE    7777   mise à jour
                                  RP
```


DEBUT
Sorties Entrées Varia Tests Ordres Transfert
50 SRA ERT ADD RA+1 1717
SM ERA EX LL 7777
(RT)9:16:=adresse mot réservé exécutif
(RA)9:16:=CCW pointer
(SM):=p, Li x
CCW count:: 128
SM EX 1XS LL 7777
NO
PK
Envoi Priorité p
FIN
SA8C ERT RP 6060
SM EAC 6060
VS
SRT EAC K 6060
51 ADD 1R
LI x := LI x +1
SZ LE 7777
SZ EX 1460
LI x :: 2^15
I 1XS RA+1
(RA)9:16 := CCWpointer+1
PK
(RA)9:16:=adresse mot réservé exéc.
SRT ERA SEQ
52 ADD RA+1
+1 → CCW count
+2 → CCW pointer
SM EX LL 6003
SZ EX 0360
SM EY 1717
SZ EX ADD 7777
EY
SZ LE K 7777
type I/O =
Input caract. 1
Input caract. 2
Input caract 3
Output caract 1
Output caract 2
Output caract 3
53 RA:=ni:=ni+1 input Ecr.50 51 54
54 RA:=ni input Ecr.50 51 55
55 RA:=ni input Ecr.50 51 53
56 RA:=ni:=ni+1 output Ecr.50 51 57
57 RA:= ni output Ecr. 50 51 58
58 RA := ni output Ecr. 50 51 56
FIN

<u>Output</u> 1er caractère

```
56                  ADD        1R
       SM    EX                LL     7777
       SZ    ERA               LE     7777
       SM    VS                LL     6003
       SRT   ERA               SMF    1717
          50       51      57
       SRT                     LE     7777
                               RP
```

2e et 3e caractère

```
57     SM    ERA   ADD         LL     1717
- 58   SM    VS                LL     14 ← 57
                                      03
                                      03 ← 58
       SRT   ERA               SMF    17 17
          50       51               58 ← 57
                                    56 ← 58
       SRT                     LE     7777
                               RP
                               PK
```

II. 3.1.2. Remarques et Commentaires.

Nous remarquerons l'usage fait ici de l'écriture dans le mot réservé exécutif. Nous avons en quelque sorte un automate à trois états :

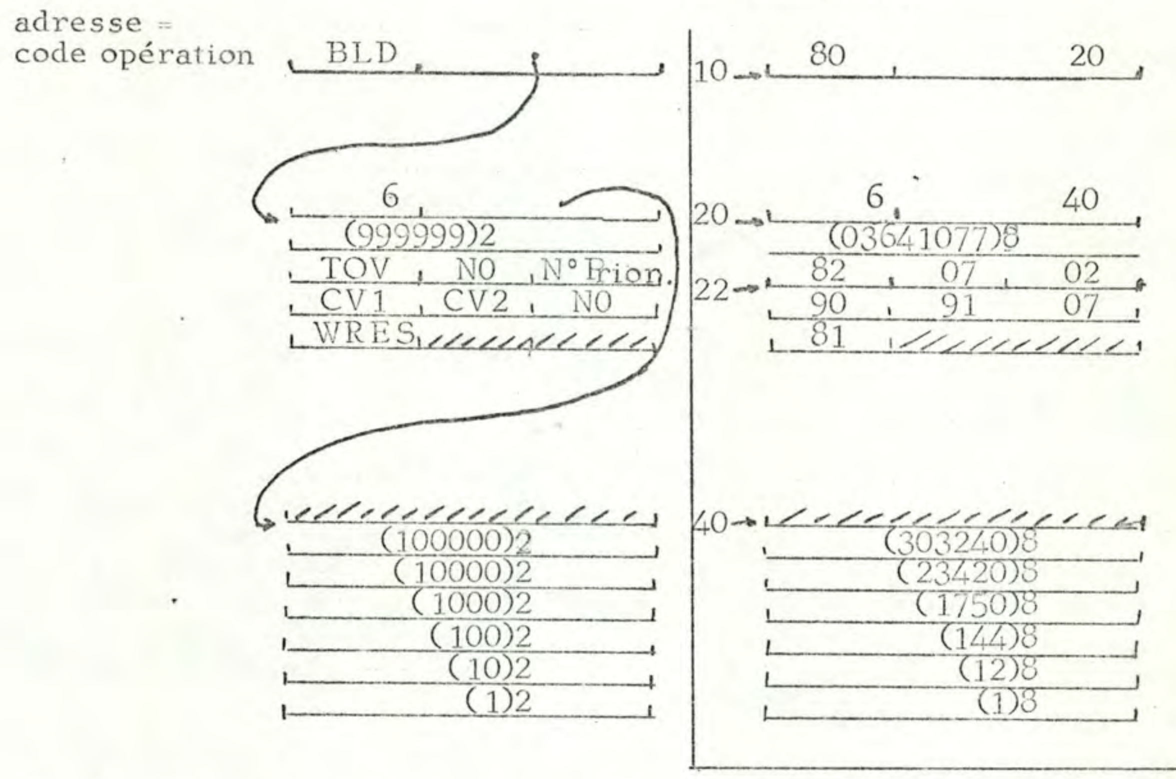
1. préparé à recevoir un caractère à cadrer à gauche.
2. préparé à recevoir un caractère à cadrer au milieu.
3. préparé à recevoir un caractère à cadrer à droite.

De même, une constante placée en Mémoire Rapide nous délivre immédiatement (par l'ordre SMF) la ligne micro à charger dans le mot réservé exécutif.

L'usage de DM comme registre de manoeuvre peut surprendre. On constatera cependant que, puisqu'à partir du st0 de T3 DM n'est plus utilisé pour définir les codes des micro instructions, il peut être utilisé par le micro-programmeur.

En cas de chaînage, quand il faut incrémenter, le CCW COUNT et le CCW POINTER, on constate que n'ayant pas donné l ordre RP, il suffit de donner l'ordre K pour recommencer l'exécution de la ligne.

II. 3.2. Conversion binaire-BCD.



BLD (80) Branch and Load champ adresse n dans la ligne-micro

Cette micro-instruction charge les registres R1 et R2 avec le contenu de la Mémoire d'adresse n et n + 1 et réalise un branchement micro-programme à l'adresse n + 2

TOV (82) Test overflow champ priorité p dans la ligne micro

Cette micro-instruction compare les registres A et R2. Si (A) ≤ (R2), il y a passage à la micro-instruction suivante; sinon :

- recherche nouvelle instruction (modification de CM),
- envoi priorité p,
- passage à la micro-instruction suivante

WRES (81) Write Result

Cette micro-instruction écrit en Mémoire à l'adresse donnée dans RI le contenu du registre R2. Elle exécute ensuite la Recherche Nouvelle Instruction.

CV1 (90) et CV2 (91) assurent la conversion (A)bin ⟶ (R2) BCD.

II. 3.2.1. Organigramme

(R1)1:8 := 6
(R1)9:16 := adresse table de conversion - 1
(A)1:24 := valeur à convertir
code micro CV1 ⩾ 10

DEBUT

nb digits := nbdigits - 1

nb digits ::0 — NON / OUI

NON: (CM):=(CM)-1 ; switch := 1

OUI: switch :=0

adt:=adt + 1
(SM)=(MPadt)

(RA)1:4 :=0
(X):= (A)

CV1

(Z):=(X)-(SM)

(X)::(SM) — < / ⩾

<: (A) := (X) ; (R2):=(R2)x16 + (RA)1:4

⩾: (X) := (Z) ; (RA):=(RA)+1

CV2

NO

switch = ? — 1 / 0

Temps 0

FIN

(A) 1:24 := 0
(R2) := résultat de la conversion.

II. 3.2.2. Microprogrammation.

CV1

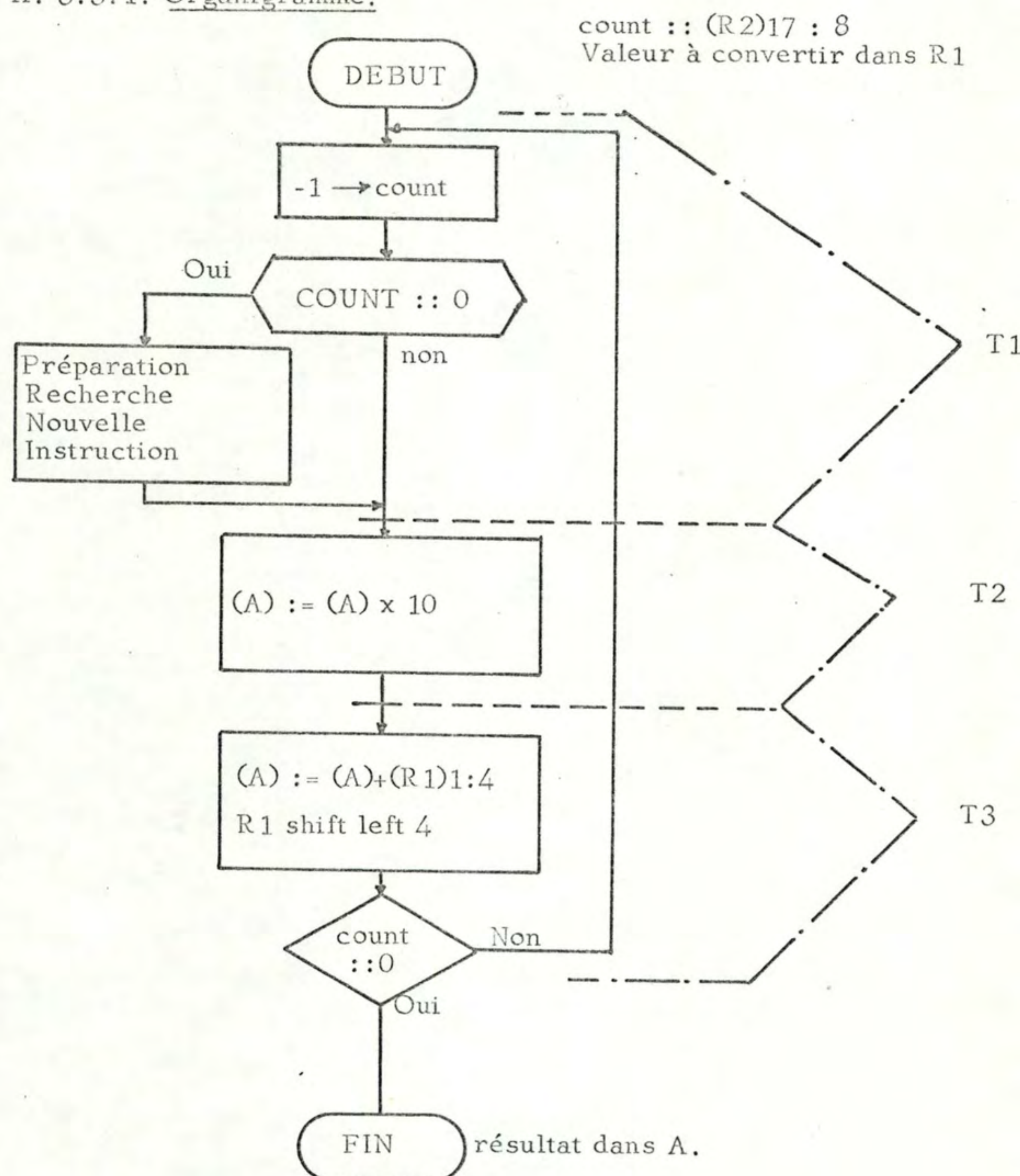
```
SR1 EX   SST1                6003   - 1 sur le nombre de digits
SZ  ER1        TZOS          0360   nombre de digits = 0 ?
SRA ECM                      1717   non recommencer la ligne micr
SR1 ERA                      7777   oui adresse table de conversio
SA  EX              RA+1     7777   (A) = valeur à convertir
SRA ER1             LL       1717   lecture table et sauvetage
                                    adresse
    ERA                      0101   (RA)21 : 4 := 0
SDM EAC                      6003   bouclage sur micro suivante.
```

CV2

```
SM  EY   SST             1R  7777
         I     DS
SZ  EX         DPK  RA+1     7777   La soustraction a été possible
    EY                       7777
SZ  EA                       7777   (A) := reste à convertir
SR2 ER2                      3776   Shift left 4
SRA ER2             K        0101   Chargement du digit calculé
```

II. 3.3. Conversion B C D-binaire sans table.

II. 3.3.1. Organigramme.



II.3.3.2. Microprogramme.

```
        SR2   EX    SST1               0303
        SZ    ER2          TZOS        0303  - 1 sur le nbr. de digits
        SRA   ECM               PK     1717  nbr. digits ≠ 0
        SCP   ERA               DMPI   1717  Recherche Nouvelle Instruction
T1      SM    ERI               LL     7777
        SM    ECM               TZ     6003
                                RA+1
        SRA   ECP               PK     1717

                    ADD
        SA    EY    EX                 7777  X = A    Y = A    Z = 2A
        SZ    EY    EX                 7777  X = 2A   Y = 2A   Z = 4A
        SZ    EX                       7777  X = 4A   Y = 2A   Z = 6A
T2      SA    EY    ADD                7777  X = 4A   Y = A    Z = 5A
        SZ    EY    EX                 7777  X = 5A   Y = 5A   Z = 10A
        SZ    EA                PK     7777           A = 10A

        SR1   EX    ADD                6003
        SZ    EX                       0201  X = 4 bits de gauche de R1
T3      SA    EY                       7777
        SZ    EA                       7777  A = A + 4 bits de gauche de R1
        SR1   ER1               PK     3776
```

II. 4. PREMIERES CONCLUSIONS.

En étendant la technique illustrée précédemment, nous constatons que :

- l'exécution d'une instruction est découpée en unités sémantiques, nommables, définies par une séquence de 8 micro-fonctions et adressables par le code de la micro-instruction.

- ces unités sémantiques communes à plusieurs instructions sont chaînées au niveau de la ligne - micro.

- la première unité est souvent un calcul d'adresse (IDXI, IDX, I) et la dernière un lien à l'instruction suivante (RNI).

Nous pouvons décrire ce mécanisme à partir de la notion de mini-langage (celui des lignes micro) assurant la séquence des différentes unités de traitement.
Le mini-langage supprime le contrôle de l'adresse au niveau micro-langage et permet une densité très élevée dans la définition d'instructions complexes chaînant des unités de traitement déjà écrites.

II. 4.1. Dynamisme au niveau mini-langage.

Etant donné un ensemble d'unités de traitement (les 256 micro-instructions) et un jeu d'instructions, nous pourrions définir de nouvelles instructions plus complexes en chaînant leurs séquences d'unités de traitement. Notre mémoire de mini-langage étant inscriptible et non bornée (la limite entre la zone mini-langage et software est purement logique) nous pouvons réaliser des extensions du jeu d'instructions sans difficultés insurmontables ...

Rappelons que ce mécanisme est rentable parce qu'il a été prévu dans le hardware. En effet, si l'EPRON n'est pas conçu en fonction d'un Set d'Instructions bien précis, il est par contre construit sur une structure bien définie à trois niveaux permettant cette construction dynamique au niveau mini-langage.

Le chapitre III nous donnera une approche du mécanisme de création de nouvelles instructions par cette méthode de chaînage d'unités de traitement. Cette création pouvant se faire "at RUN TIME" ou "at COMPILE TIME".

Ce dynamisme permet de compléter le jeu d'instructions.
Le mécanisme suivant permet, lui, de compléter ou modifier le décor virtuel.

II. 4.2. Dynamisme au niveau micro-langage.

Quelques exemples vont nous permettre d'entrevoir un outil assez prodigieux.

II. 4.2.1. RNI Statistiques.

Soit la micro-instruction RNI écrite comme suit :

```
SCP  ERA  ADD        DMPI  1717
SM   ERI             LL    7777
SM   ECM             RA+1  6003
SRA  ECP             SMF   1717
     constante adresse table des compteurs
SRT  ERA                   7777
SM   ERA             SEQ   6003  RA pointe vers le comp-
                                 teur correspondant au
                                 code opération.
          ADD        1R
SM   EX              LL    7777
SZ              DS   LE    7777  Incrémentation du
                     K           compteur correspondant
                     SMF         au code opération
     constante n° priorité
SRT  EAC                   6060  Envoi priorité en cas
     VS              K           d'overflow
```

En écrivant ces 15 mots en mémoire rapide, nous allons pouvoir faire des statistiques précises sur le programme sans le perturber, sans devoir le réécrire et sans freiner trop son exécution.

En effet, à chaque exécution d'instruction, un compteur déterminé par son code opération est incrémenté de 1 et déclenche une interruption si il y a overflow de ce compteur.

II. 4.2.2. RNI Trace.

Soient les 2 micro-instructions RNICP et RNICO.

```
RNICP   SCP  ERA  SST            DMPI  1717
        SM   ERI  EX             LL    1717
        SM   ECM                 RA+1  6003
        SRA  ECP                 SMF   1717
             Constante = Valeur à comparer au CP
        SRT  EY                        1717
        SRT  EAC        TZOS           6060
             VS                  K


RNICO   SCP  ERA  SST            DMPI  1717
        SM   ERI                 LL    7777
        SM   ECM  EX             RA+1  6003
        SRA  ECP                 SMF   1717
             Constante = Valeur à comparer au C.O.
        SRT  EY                        0303
        SRT  EAC        TZOS           6060
             VS                  K
```

Elles envoient une priorité si :

RNICP Le CP est égal à une valeur définie.

RNICO Le code opération est égal à une valeur définie.

De nouveau, sans rien modifier au programme, on peut demander un arrêt sur une instruction déterminée soit par son adresse, soit par son code opération.

N.B. : Un ordre Figeage peut remplacer l'ordre d'envoi d'une priorité.

II. 4.2.3. Conditions d'une application optimale.

Ces trois exemples nous ont montré comment en transformant une unité de traitement utilisée dans toutes les instructions on peut modifier complètement l'aspect du système.
On peut de même, en modifiant les micro-instructions d'adressage, par exemple, passer de 16 à 256 registres et un déplacement de 4 K à 256 mots.

Cette méthode impose cependant de :

- découper proprement les unités de traitement; il faut que les micro-instructions représentent un réel traitement standard dont l'input et l'output soient toujours définis.

- utiliser systématiquement ces unités de traitement si l'opération recherchée l'inclu, sinon (pour optimiser) noter clairement dans un tableau la ou les fonctions que réalise cette micro pour pouvoir écrire l'équivalente pour le nouveau mécanisme.

- prévoir de la place en MR "derrière" les micro susceptibles d'être modifiées ou complétées. L'ordre SEQ permet en effet de remplacer 1 micro par 1 ou plusieurs.

Le lecteur pourrait traiter le cas suivant :

Dans les algorithmes scientifiques il est souvent nécessaire de travailler avec une grande précision lorsqu'on est proche de la solution alors que les premières phases peuvent être beaucoup plus grossières.

Imaginer une architecture globale capable d'affiner dynamiquement la précision.

- Le programme est fixe,
- Le jeu d'instructions comprend une instruction spéciale PREC, n

Suivant la quotation de Knuth, cet exercice a les quotes : (1)

25 si on l'imagine simplement

38 si on le définit jusqu'au bout

45 si implémenté, le système tourne correctement.

---

(1) Knuth "The art of computer programming" Vol. 1 Fundamental Algorithms pp. XVII - XIX.

CHAPITRE III

DEFINITION D'UNE ARCHITECTURE SOFTWARE FIRMWARE DYNAMIQUE.

1. OBJECTIF.
1.1. Hypothèses.
1.2. Propositions.

2. METHODES ET MECANISMES.
2.1. Division en niveaux.
2.1.1. Langage Intermédiaire.
2.1.2. Communication entre niveaux.
2.2. Mécanisme d'appel.
2.2.1. Type a.
2.2.2. Type b.

3. REALISATION.
3.1. Instruction Assemble.
3.1.1. Description du source.
3.1.1.1. Flags.
3.1.1.2. Instructions.
3.1.1.3. Exemple.
3.1.2. Table des micro-programmes.
3.1.3. Mécanisme des branchements internes.
3.1.3.1. Déclaration d'un label.
3.1.3.2. Branchement à un label.
3.1.3.3. Conversion Label adresse.
3.1.4. Organigramme détaillé de l'instruction AS, n
3.1.5. Sous-micro-programmes utilisés.
3.1.5.1. Put μ .
3.1.5.2. GET μ .
3.2. Instruction LM et LME.
3.2.1. RMI modifié.
3.2.2. LM type a.
3.2.3. LME type b.
3.2.4. Protection des Macros existantes.
3.2.5. Gestion des Tables.
3.3. Bilans.
3.3.1. Exemple implémenté.
3.3.1.1. Méthode classique.
3.3.1.2. Type a.
3.3.1.3. Type b
3.3.2. Bilan place mémoire.
3.3.3. Bilan temps d'exécution.

CHAPITRE III

DEFINITION D'UNE ARCHITECTURE SOFTWARE - FIRMWARE DYNAMIQUE

Les deux chapitres précédents nous ont introduits aux mini- et micro-langages. Cette étude nous a mis en face d'une véritable architecture dotée de toutes les primitives d'un langage de Base. De plus, le parallélisme dans les actions ainsi qu'une utilisation plus directe du hardware nous ont fait pressentir la puissance de ces niveaux mini et micro.

D'autre part, travailler uniquement à ces deux niveaux serait coûteux en place mémoire et en temps de programmation. Nous voudrions proposer un mécanisme facile à utiliser et permettant au programmeur de bénéficier de la puissance et de la souplesse des niveaux inférieurs.

Le macro assembleur supporté par firmware décrit ici, n'est qu'un point parmi d'autres dans la définition d'une architecture software-firmware dynamique. Nous avons centré notre attention sur ce mécanisme pour qu'en le développant le plus possible (dans le cadre de ce travail) nous puissions en tirer déjà quelques conclusions quantitatives. L'interaction firmware-software ne se réduit pas à ce macro-assembleur et encore moins aux options précises implémentées ici.

- - - - - - - -

III. 1. OBJECTIF.

Très simplement exprimé, notre objectif est "plus d'efficience" c'est-à-dire soit un gain en place mémoire, soit un gain en temps, éventuellement un gain sur les deux volets.

III. 1.1. Hypothèses.

Parmi les mécanismes appliqués en programmation, nous voyons deux recherches menées parallèlement :

- l'utilisation de procédure pour gagner de la place mémoire ①
- l'utilisation de macro pour ne pas perdre de temps par les mécanismes d'appel tout en facilitant l'écriture d'unités de traitement ②.

Sur une machine micro-programmée, une instruction est en fait un appel de procédure micro-programme. Cet appel est très efficient puisque l'architecture prévue à cet effet, associe des entités hardware différentes pour le processus appelant (niveau langage software) et appelé (niveau mini-langage). ③

III. 1.2. Propositions:

L'énoncé simultané des points ①, ② et ③ conduit naturellement aux mécanismes suivants :

1. un compilateur de Macros générant du micro-programme.
2. un mécanisme d'appel de procédures micro-programme.

Grâce à ces deux mécanismes,nous réaliserons ainsi un gain de place mémoire (nous avons une seule procédure micro-programmée et non l'expansion répétée d'une macro) sans perte de temps puisque l'appel de procédure est assité par le hardware. (1)

---

(1) Nous verrons même que dans plusieurs cas un gain de temps peut être réalisé par la suppression des temps de "Fetch d'instruction".

III. 2. Méthodes et Mécanismes.

Deux points de vue doivent être envisagés ici, la facilité de l'utilisateur du mécanisme et l'efficience du mécanisme produit. Comme toujours nous avons dû choisir un compromis.

Notre mécanisme veut présenter au programmeur l'aspect d'un macro-assembleur classique. Le langage source du compilateur de micro-programmes sera donc le langage d'assemblage lui-même. L'appel de la procédure micro-programme sera généré à la rencontre de l'application de la Macro.

III. 2.1. Division en niveaux.

A quel niveau allons nous écrire le compilateur ?

En langage de base? En mini-langage ?

Nous ne justifions pas directement l'importance de cette question. Au fil du développement de ce mécanisme, nous devinerons l'enjeu des options prises.

III. 2.1.1. Langage Intermédiaire.

Nous avons défini un langage intermédiaire, produit par le software et utilisé comme source par le compilateur micro-programmé. Ce langage intermédiaire pourra éventuellement être écrit directement par un programmeur plus soucieux d'optimisation. Il sera le plus souvent produit par le macro assembleur après la phase d'assemblage conditionnel.

Plusieurs zones de test sont introduites dans ce langage intermédiaire pour éviter au maximum les erreurs et avoir un interface propre entre le software et le compilateur micro-programmé.

III. 2.1.2. Communication entre niveaux.

Le compilateur micro-programmé est appelé par une instruction AS (Assemble). Bien que pour l'instant nous ne voyons pas directement l'usage de cette instruction "at RUN-TIME", cela est possible. L'emploi "at DESIGN TIME" et "at COMPILE TIME" sera plus habituel.

III. 2.2. Mécanisme d'appel.

Nous distinguerons deux types de Macro, suivant le format de l'application.

III. 2.2.1. Type a.

La macro définie garde le format des instructions mais peut avoir un complément d'information sur les opérandes dans les mots suivants.

Le nombre de telles macros est réduit à 16 par le fait que le code de la Macro est défini sur un nombre restreint de bits. On considère ici que 16 instructions sont variables et du type MACRO.

III. 2.2.2. Type b.

La Macro définie a un code opération spécial (Code Macro Etendue) sur 6 bits et un n° d'identification dans la zone opérande.

Le nombre de telles macro est à priori "illimité" (2^{18}).

Pour plus de facilités nous le fixerons cependant à 256.

Avant de pouvoir être utilisée une macro doit être "armée". Les instructions LM (Load Macro) et LME (Load Macro Etendue) assignent une adresse micro-programme à un code MACRO.

III. 3. REALISATION.

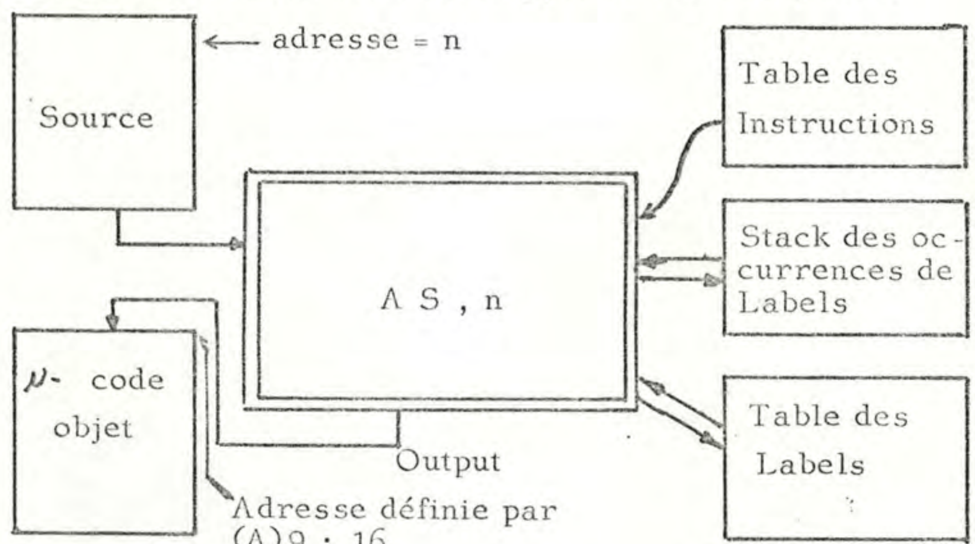
Le software réalise une première conversion de la définition et de l'application des Macros. Nous ne développons pas ce travail ici.

Le firmware implante le mécanisme en deux phases indépendantes :

- la création d'un micro-code à partir du source par l'instruction A.S(Assemble).
- l'assignation d'une adresse micro-programme à un code instruction par les Instructions LM (Load Macro) et LME (Load Macro Etendue).

III. 3.1. Instruction Assemble.

Nous avons vu précédemment que la "compilation" d'une Macro en micro-programmes se réalise en deux phases, l'une sous contrôle du software, l'autre sous contrôle du firmware.
Nous nous attachons ici à développer la seconde phase.



L'instruction AS (Assemble) réalise cette phase de compilation du source en micro-code.Elle fait appel à une table donnant p our chaque instruction le micro-code associé à générer et dispose de deux zones manoeuvre, un stack des occurrences de labels et une table de labels.

Nous détaillerons rapidement le format et les caractéristiques de chacune des zones suivantes :

-Source
-Table des Instructions
-Stack des occurrences de Labels et Table des Labels.
-Micro-code objet.

III. 3.1.1. Description du Source.

Il est composé d'une séquence de Flags et d'instructions (chacun(e) sur un mot de 24 bits).

Les Flags indiquent au compilateur les opérations à effectuer ainsi que les valeurs des paramètres des différentes instructions.

Dans les pages qui suivent, nous centrons notre attention sur le mécanisme. Pour ne pas compliquer la présentation, nous ne précisons pas les méthodes d'adressage (indirect-index).

III. 3.1.1.1. Flags.

Ils sont divisés en 8 types distingués d'après les bits 6 à 8. Pour tous, les bits 1 à 5 doivent être à 0. Les autres bits indiquent 1 ou plusieurs paramètres. Si ils ne sont pas significatifs ils doivent être à 0.

Ces 8 types sont :

- Global long,	les bits 9 à 24 contiennent la valeur de ce global utilisé dans l instruction suivante.
- Global court,	les bits 17 à 24 contiennent la valeur de ce global utilisé dans l'instruction suivante.
- Paramètre,	l'instruction suivante utilise un paramètre de la MACRO. Les bits 17 à 24 indiquent le n° de la ligne (-1) où se trouve le paramètre dans l'application de la MACRO. Les bits 9 à 16 précisent le Type de cadrage des paramètres.
- NO ,	l'instruction suivante utilise le même paramètre que la précédente ou n'en utilise pas.
- MEXIT,	ce flag demande la génération d'un Return; c'est-à-dire incrémentation du P compteur de LM et Recherche Nouvelle Instruction. LM (Longueur MACRO) est égal aux nombres de mots (-1) de l'application de la MACRO.
- Alignement, - Branch Interne	définis plus loin en même temps que la Table des Labels et le Stack des occurrences de Labels. Les bits 21 à 24 précisent le Label référé
- MEND,	signale la fin du Source.

FLAGS.

Format	Flag	Champ	Description
0 0 , X X , X X	Global long		
0 1 , 0 0 , X X	Global Court		
0 2 , Type C , Numéro	Paramètre	Type C Numéro	Type de cadrage Numéro de ligne où se trouve l'opérande dans l'application de la Macro.
0 3 , 0 0 , 0 0	NO		
0 4 , 0 0 , L M	MEXIT	LM	Longueur Macro = nombre de mots que prend l'application de la Macro.
0 5 , 0 0 , Label	Alignement		
0 6 , 0 0 , Label	Branch Interne	Label	Etiquette (4 bits) interne à la Macro.
0 7 , 0 0 , 0 0	MEND		

III. 3.1.1.2. Instructions.

Pour rappel, dans notre architecture, une instruction est définie par un code opération de 6 bits et un mode d'adressage de 2 bits.

Ces 8 bits seront cadrés à droite dans le mot. Les 16 bits de gauche doivent être à 1.

Un exemple simple est donné ci-dessous. Chacune des représentations successives est décrite.

III. 3.1.1.3. Exemple.

Def. de la Macro.

Cette macro a pour objet l'addition du 1er paramètre au second ou au troisième suivant le signe du 1er.

```
        MACRO
        ADD     &A,&B,&C
        L       &A
        BN      &LAB          Branch si négatif
        AD      &B          ⎫
        ST      &B          ⎭ Addition sur B
        MEXIT
&LAB    AD      &C          ⎫
        ST      &C          ⎭ Addition sur C
        MEND
```

Application de la MACRO.

```
ADD     A, B, C
```

Génération lors de l'expansion de la MACRO par un mécanisme classique.

```
2300    L       A
2301    BN      2305
2302    AD      B
2303    ST      B
2304    B       2301
2305    AD      C
2306    ST      C
2307    ....
```

La transformation par le software de la Définition de la Macro donne le Source suivant à l'intention de l'instruction AS, n

si l'application
est de type a

Source		Code		
ADD, A	—n→	0 3, 0 0 0 0	FLAG	NO
////// B		L		
////// C		0 6, 0 0 0 1	FLAG	Branch Interne
		B N		
		0 2, T C 0 0	FLAG	Paramètre B
		A D		
		0 3, 0 0 0 0	FLAG	NO
		S T		
		0 4, 0 0 0 2	FLAG	MEXIT
		0 5, 0 0 0 1	FLAG	Alignement
		0 2, T C 0 1	FLAG	Paramètre C
		A D		
		0 3, 0 0 0 0	FLAG	NO
		S T		
		0 4, 0 0 0 2	FLAG	MEXIT
		0 7, 0 0 0 0	FLAG	MEND

si l'opération
est de type b

Source		Code		
MAC, ADD, //////	—n→	0 2 T C 0 0	FLAG	Paramètre A
////// A		L		
////// B		0 6 0 0 0 1	FLAG	Branch Interne
////// C		B N		
		0 2 T C 0 1	FLAG	Paramètre B
		A D		
		0 3 0 0 0 0	FLAG	NO
		S T		
		0 4 0 0 0 3	FLAG	MEXIT
		0 5 0 0 0 1	FLAG	Alignement
		0 2 T C 0 2	FLAG	Paramètre C
		A D		
		0 3 0 0 0 0	FLAG	NO
		S T		
		0 4 0 0 0 3	FLAG	MEXIT
		0 7 0 0 0 0	FLAG	MEND

III. 3.1.2. Table des micro-programmes.

Cette table à 64 entrées, définit le type d'assemblage à réaliser pour chaque code instruction, et le contenu à assembler.

Les bits 5 à 8 de chaque entrée définissent le type.

A

//////	0000	//////	micro
1 4	5 8	9 16	17 24

l'instruction est réalisée par 1 seule micro-instruction les bits 17 à 24 donnent alors le coding de cette micro.

B

//////	0001	micro 1	micro 2

l'instruction est réalisée par 2 micro-instructions consécutives et placées sur 1 même mot
les bits 9 à 16 et 17 à 24 donnent le coding de ces deux micro.

C

//////	0010	micro 1	micro 2

Comme le type B mais les micro-peuvent être implantées sur deux lignes consécutives.

D

L	0011	ad

micro 1	micro 2	micro 3
micro 4	micro 5	...

l'instruction est réalisée par 3 ou plus de 3 micro-instructions. Un code complémentaire précise :

L : nombre de micro à implanter (bits 1 à 4)

ad : adresse de la zone ou l'on trouve le coding de cette séquence de micro (bits 9 à 24).

III. 3.1.3. Mécanisme des branchements internes.

Une instruction de branchement interne à la macro doit être implémentée tout différemment d'un branchement . L'un modifie le compteur micro-programme, l'autre le compteur programme avec recherche nouvelle instruction.

Un Flag permet de signaler ainsi les Branch Internes. D'autre part, puisque les adresses micro-programmes sont inaccessibles au niveau software, il faut prévoir une conversion entre des labels et des adresses micro-programmes. Cette conversion menée grâce à la table des labels et le stack des occurrences de labels assure de plus la validité du branchement par le flag Alignement.

III. 3.1.3.1. Déclaration d'un Label.

Le Flag d'alignement réalise 2 opérations :

- aligner la liste de code objet sur la frontière d'un mot.
- garnir la table des labels.
  A l'entrée correspondant au numéro du label on charge la valeur de (A)9:16, pointeur vers le sommet de la liste du code objet.

III. 3.1.3.2. Branchement à un Label.

Le Branch Interne génère 2 mots dans la liste du code objet et ajoute une adresse au stack des instructions de Branch Interne.

- Les deux mots générés auront p our effet de :
  1. charger RI de la valeur = adresse de branchement.
  2. tester les conditions de branchement et effectuer le branchement.
- L'adresse chargée au sommet du stack permettra, lors de la rencontre du FLAG MEND, d'aller remplacer le label par l'adresse correspondante du code objet.

III. 3.1.3.3. Conversion label adresse.

Le remplacement des labels se fait suivant le schéma ci-dessous:

1. si le stack est vide - terminer - sinon aller en 2
2. lire champ adresse du mot au sommet du stack. Cette adresse (adbr) est celle d'une instruction de Branch Interne.

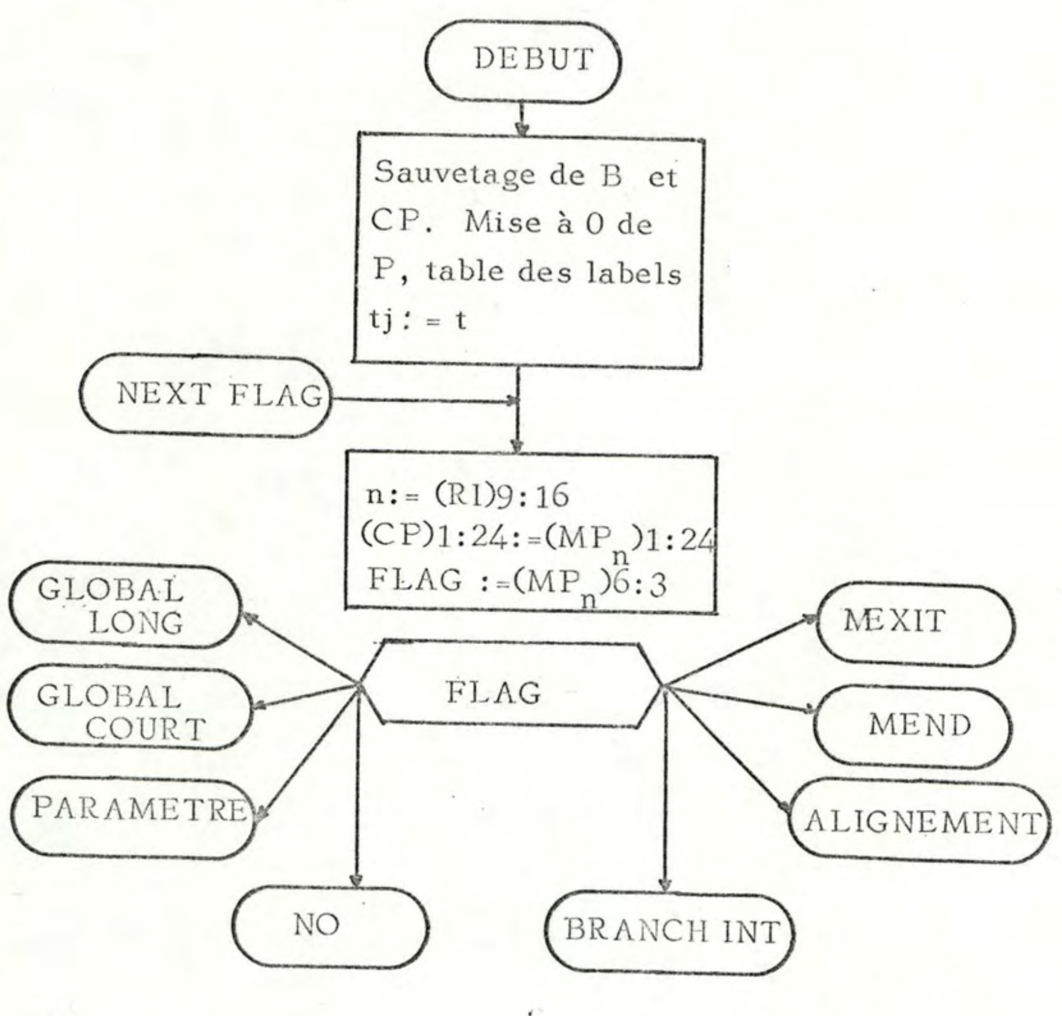
3. lire les 4 bits de droite du mot d'adresse adbr.
   Ces quatre bits indiquent le numéro du label à convertir.

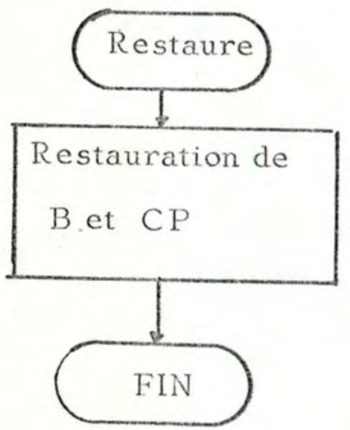
4. lire dans la Table des Labels l'adresse code objet
   correspondant à ce label.

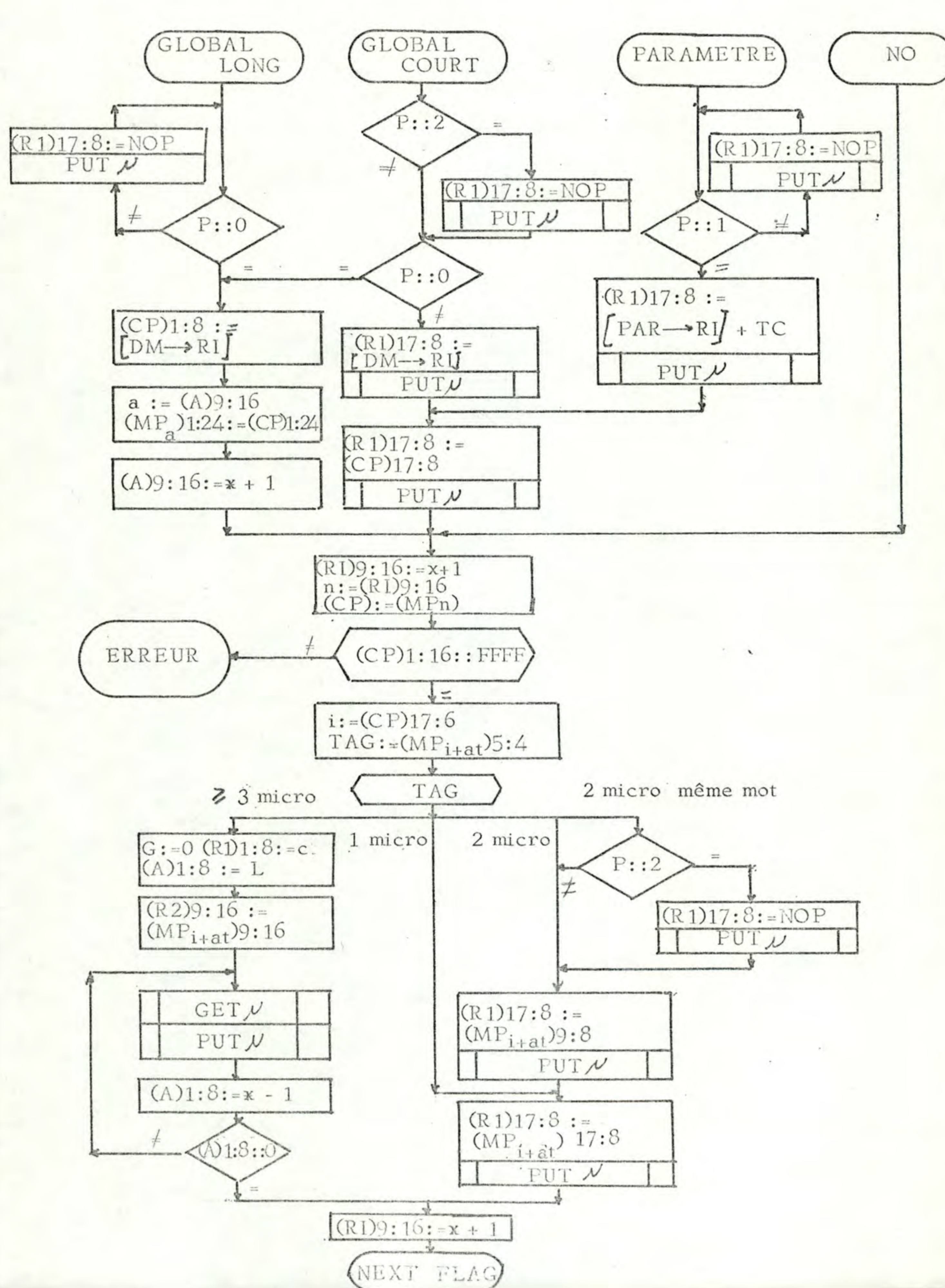
5. écrire cette adresse dans la zone adresse du mot adbr

6. diminuer de 1 la hauteur du stack - aller en 1)

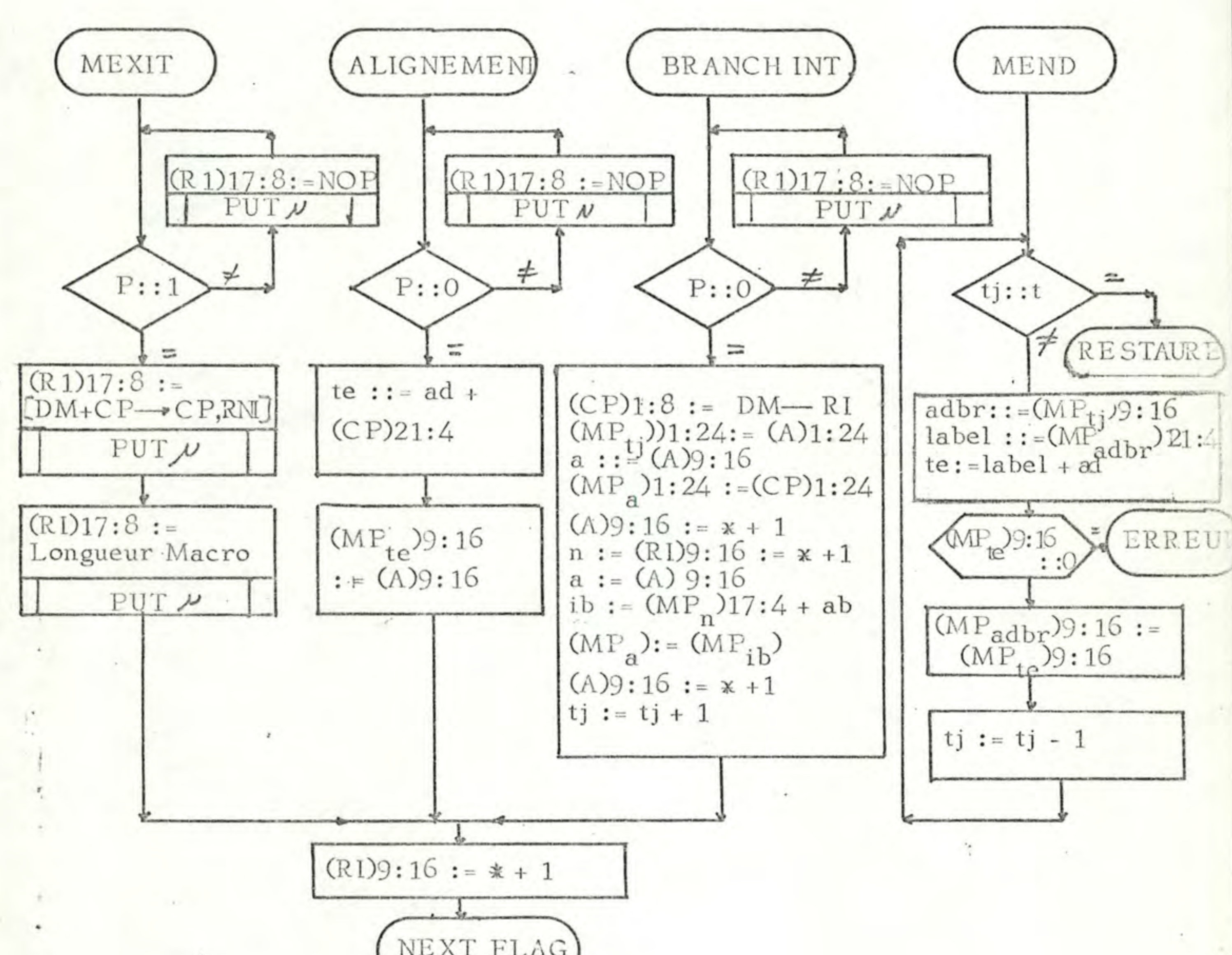
III. 3.1.4. Organigramme détaillé de l'instruction AS,n

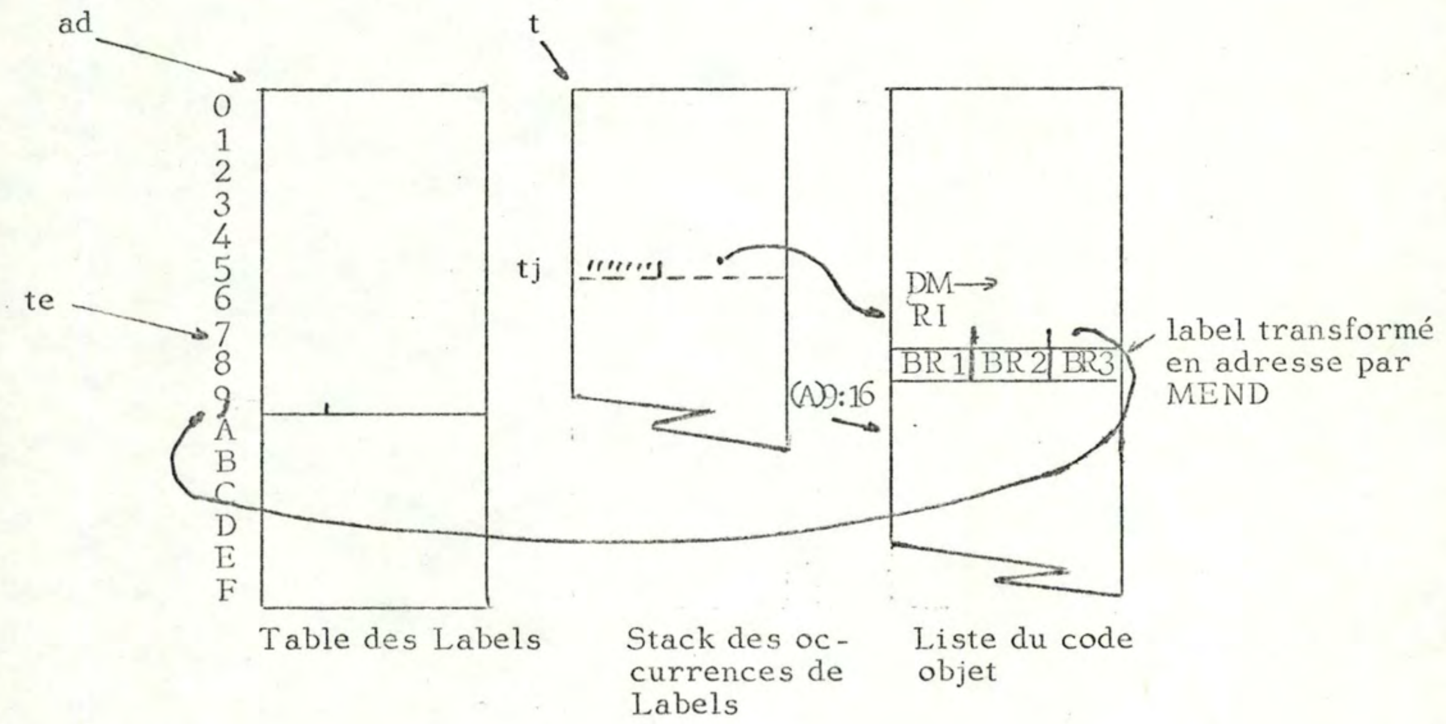






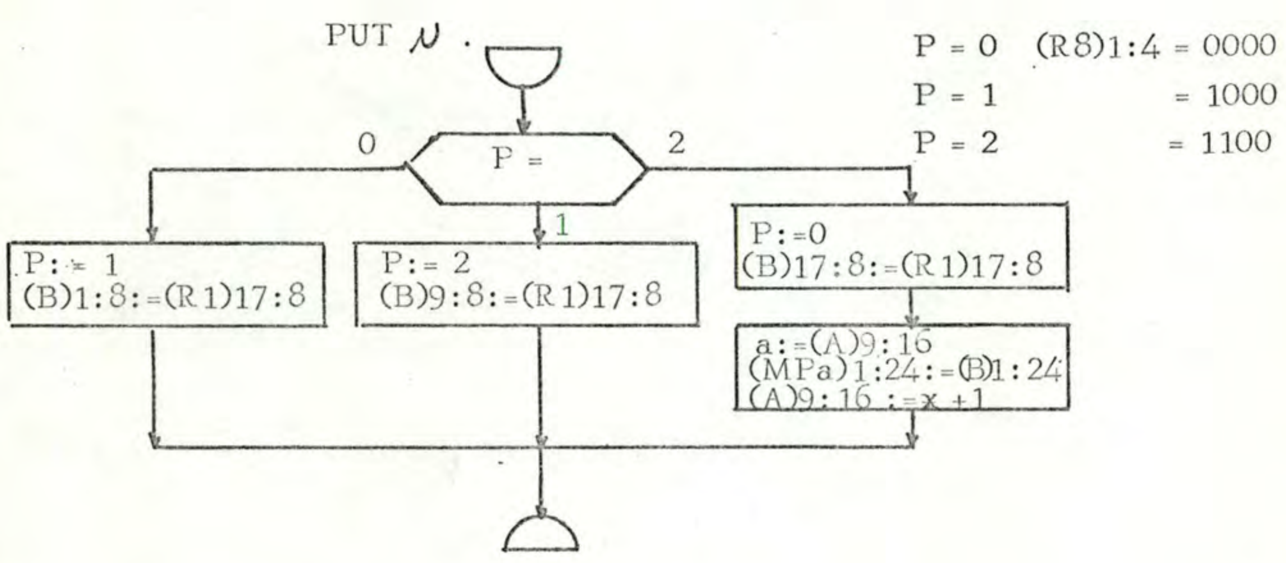
GLOBAL LONG
GLOBAL COURT
PARAMETRE
NO
(R1)17:8:=NOP
PUT ν
P::0
P::2
P::1
P::0
(CP)1:8 := [DM→RI]
(R1)17:8 := [DM→RI]
(R1)17:8 := [PAR→RI] + TC
a := (A)9:16
(MPa)1:24:=(CP)1:24
(A)9:16:=x + 1
(R1)17:8 := (CP)17:8
(RI)9:16:=x+1
n:=(RI)9:16
(CP):=(MPn)
ERREUR
(CP)1:16::FFFF
i:=(CP)17:6
TAG:=(MPi+at)5:4
TAG
≥ 3 micro
2 micro même mot
1 micro
2 micro
G:=0 (RI)1:8:=c
(A)1:8 := L
(R2)9:16 := (MPi+at)9:16
GET ν
(A)1:8:=x - 1
(A)1:8::0
(R1)17:8 := (MPi+at)9:8
(R1)17:8 := (MPi+at) 17:8
(RI)9:16:=x + 1
NEXT FLAG

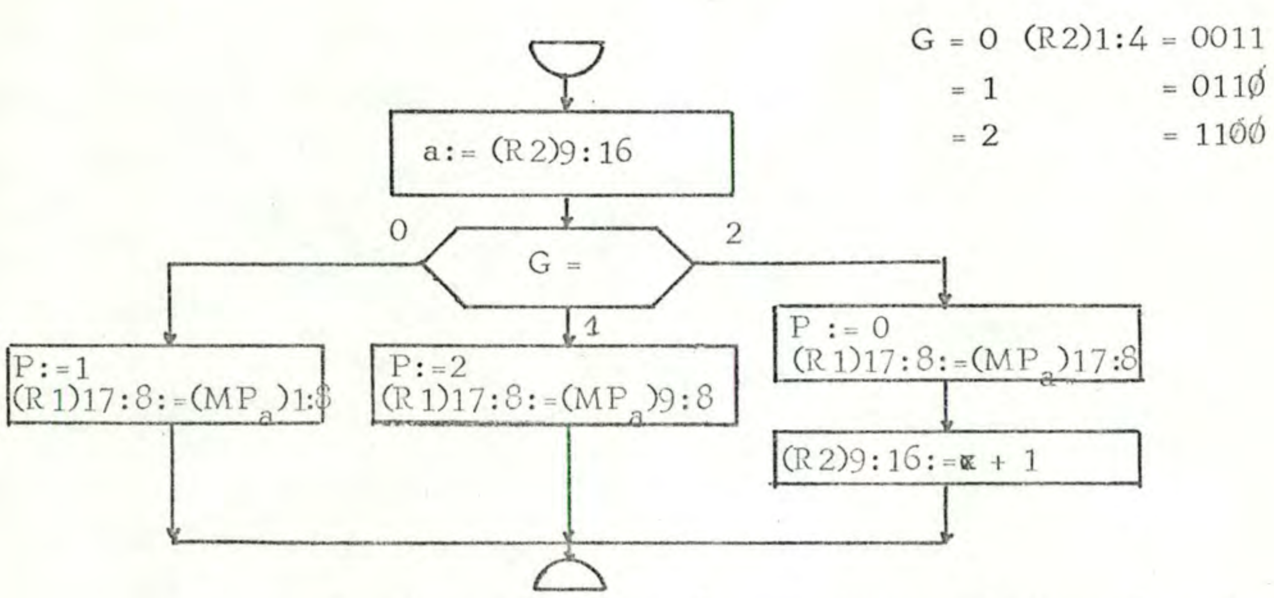

MEXIT
ALIGNEMENT
BRANCH INT
MEND
(R1)17:8:=NOP
PUT ν
P::1
≠
=
(R1)17:8 :=
[DM+CP→CP,RN]
PUT ν
(RI)17:8 :=
Longueur Macro
PUT ν
P::0
te ::= ad +
(CP)21:4
(MPte)9:16
:= (A)9:16
(CP)1:8 := DM— RI
(MPtj))1:24:= (A)1:24
a ::= (A)9:16
(MPa)1:24 :=(CP)1:24
(A)9:16 := ж + 1
n := (RI)9:16 := ж +1
a := (A) 9:16
ib := (MPn)17:4 + ab
(MPa):= (MPib)
(A)9:16 := ж +1
tj := tj + 1
tj::t
RESTAURE
adbr::=(MPtj)9:16
label ::=(MPadbr)21:4
te:=label + ad
(MPte)9:16
::0
ERREUR
(MPadbr)9:16 :=
(MPte)9:16
tj := tj - 1
(RI)9:16 := ж + 1
NEXT FLAG


ad
t
0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F
te
tj
DM→
RI
BR 1
BR 2
BR3
(A)9:16
label transformé en adresse par MEND
Table des Labels
Stack des occurrences de Labels
Liste du code objet

III. 3.1.5. Sous micro-programmes utilisés.

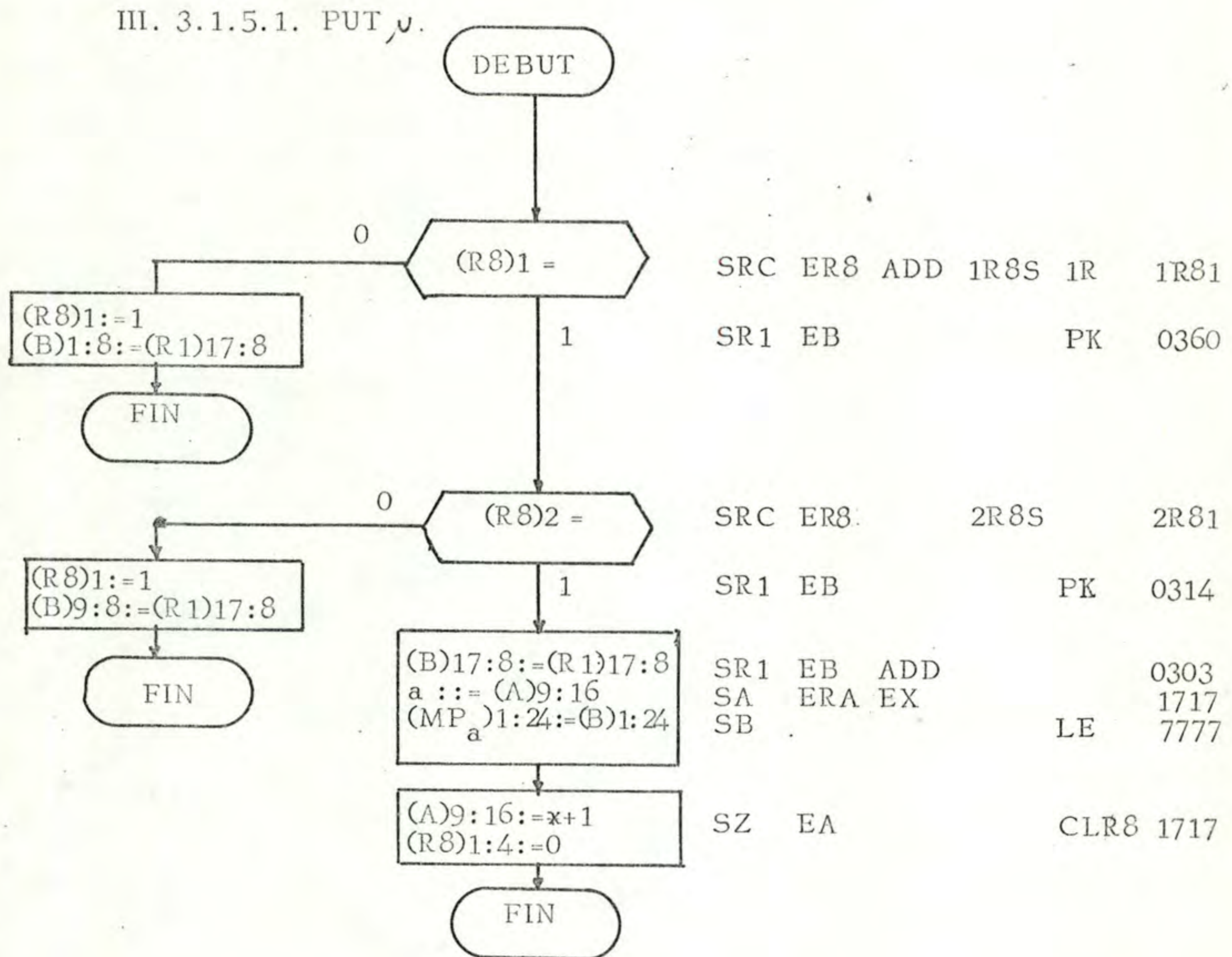
Nous n'en donnons que 2 à titre d'exemples. Ils illustrent la puissance de notre mécanisme hardware qui en 8 bits de mini-langage permet de déclencher l'appel d'une telle routine. Ces micro programmes permettent respectivement d'écrire et de lire 8 bits sur une zone mémoire. Ils mettent à jour l'adresse du mot à écrire (respectivement à lire) ainsi que le pointeur indiquant la position du byte dans le mot.





Ces deux micro-programmes correspondent aux deux ν -instructions PUT ν et GET ν . Nous les détaillons ci-dessous.

III. 3.1.5.1. PUT,υ.



(A)9:16 ::= adresse courante du code généré.

(B)1:24 ::= registre tampon.

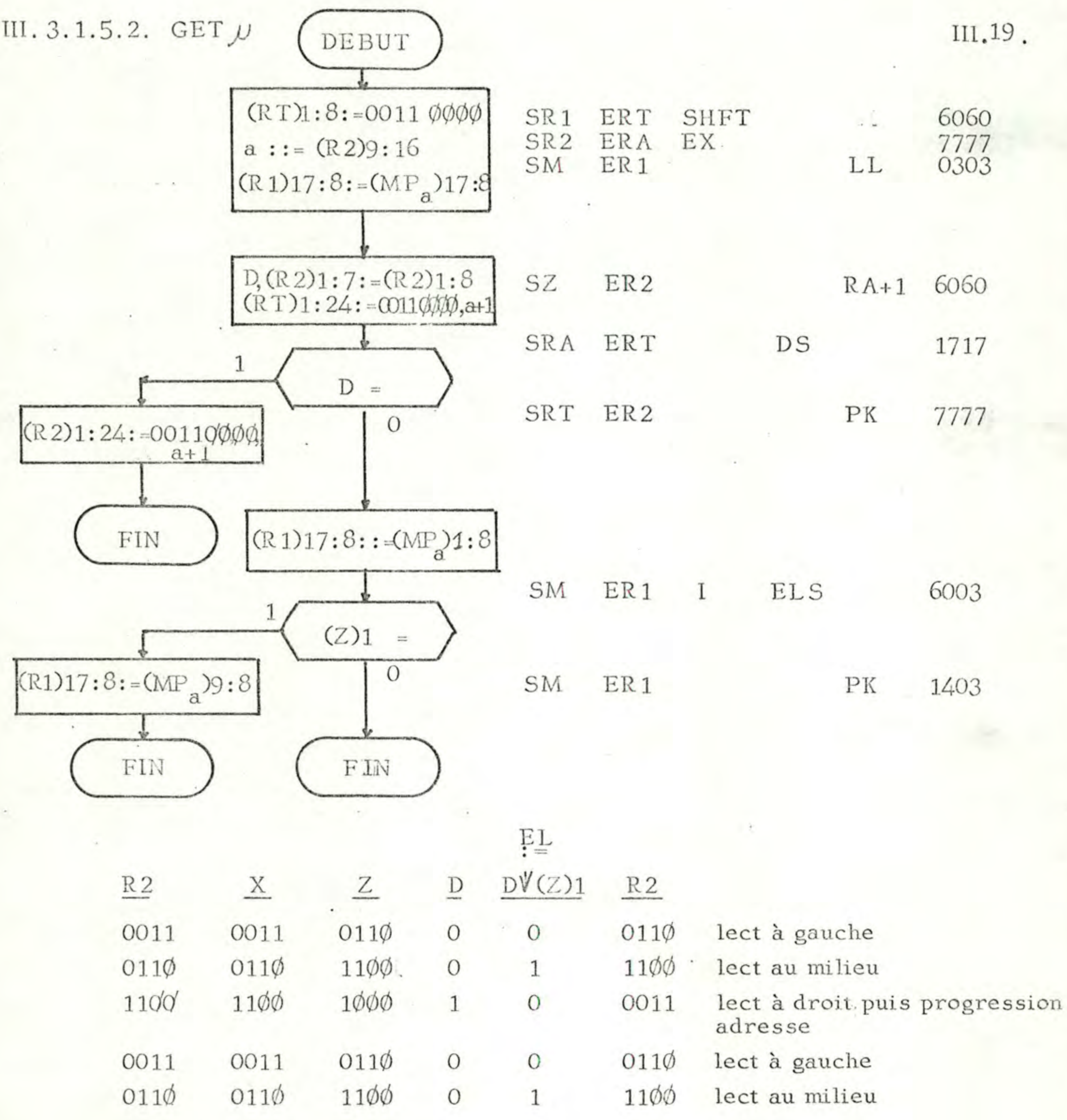
(R8)1:4 compteur module 3, évolue comme suit 0000, 1000, 1100, 0000

A chaque appel de cette micro-instruction, les 8 bits de droite de R1 sont écrits à la suite du code généré :

- la 1ère fois , ces 8 bits sont chargés dans le tampon à gauche
- la 2ème fois , ces 8 bits sont chargés dans le tampon au milieu
- la 3e fois ces 8 bits sont chargés dans le tampon à droite,

puis le tampon est écrit en Mémoire à l'adresse donnée par A. A est enfin incrémenté de 1.

III. 3.1.5.2. GET μ



R2	X	Z	D	D⊻(Z)1	R2	
0011	0011	011Ø	0	0	011Ø	lect à gauche
011Ø	011Ø	11ØØ	0	1	11ØØ	lect au milieu
11ØØ	11ØØ	1ØØØ	1	0	0011	lect à droit puis progression adresse
0011	0011	011Ø	0	0	011Ø	lect à gauche
011Ø	011Ø	11ØØ	0	1	11ØØ	lect au milieu

(R2)9:16 ::= adresse courante de la zone à lire.

(R1)1:8 ::= constante = 0011 ØØØØ

(R2)1:8 ::= compteur à décalage

(R1)17:8 ::= zone tampon

Suivant un mécanisme semblable à celui de PUT μ , à chaque appel cette micro-instruction charge 8 bits de la zone d'entrée dans le registre R1. Elle modifie un compteur modulo 3 et éventuellement le pointeur d'adresse pour assurer une prise en charge continue des différents bytes.

III.3.2. Instructions LM et LME.

Avant de décrire leur mécanisme et les tables qu'elles utilisent, nous rappelons sommairement la topologie de la Mémoire et le mécanisme de la Recherche nouvelle Instruction (cfr. chapitre II).

A la fin de l'exécution d'une instruction, une micro RNI exécute un branchement micro-programme à la ligne micro correspondant au code opération de la nouvelle instruction à exécuter.

Cette micro RNI est redéfinie ci-dessous, légèrement modifiée pour prendre en compte les MACROS ETENDUES. Si le code opération est ≠ 0, le fonctionnement est comparable à celui vu précédemment; si le code opération = 0 (code Macro Etendue), le champ des bits 9 à 16 est considéré comme un point d'entrée dans une table. Cette Table des Macros Etendues donne l'adresse du micro-programme correspondant à la Macro Etendue à exécuter.

Remarque :

Ce passage par la Table des Macros Etendues a pour but de réduire les erreurs possibles : Branchement à des adresses micro-programmes non préalablement assemblées etc....

Les Instructions AS, LM et LME étant en mode superviseur, celui-ci peut effectuer les contrôles voulus.

III. 3.2.1. RNI modifié.

DEBUT

Démasquage priorités intern.

cp:=(CP)9:16
(RI)1:24:=(MP$_{cp}$)1:24

OP:=(RI)1 : 8
OP ::0

= :

EXT := (RI)9:8
EXTENT:=EXT+256

(CM)9:16 :=
(MP$_{EXTENT}$)9:16

MACRO ETENDUE

(CP)9:16:=x +1
Branch

FIN

≠ :

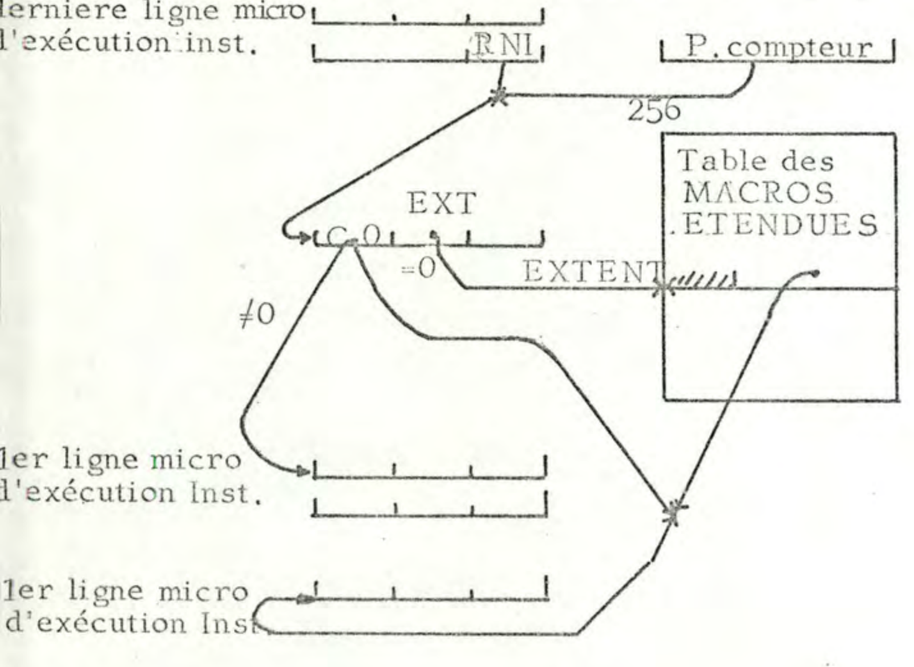
(CM)17:8:=OP
(CP)9:16:=x +1

Branch

FIN

Instruction Normale ou Macro Courte

SCP	ERA	ADD		DMPI	1717	
SM	ECM	EX		LL	6003	
SM	ERI		TZOS	RA+1	7777	
SRA	ECP			PK	1717	Instr.nor-
SRA	ECP	ADD		1R	1717	male
SM	ERA				1403	
SZ	ERA				0314	
SM	ECM			LL	1717	



III. 3.2.2. LM type a

```
|_CO_|___A___|
|____|___B___|
|____|___C___|
```

Dans le type a, le format de l'appel de la MACRO est identique à celui d'une instruction. Il y aura donc branchement micro-programme à l'adresse définie par le code Instruction.

Nous avons 17 mots (64 en mettant le traitement de l'indexage et de l'indirection en jeu) formant une table.

A chaque entrée de la table correspond un code opération de la Macro. Il suffit de garnir la zone adresse du mot avec l'adresse du micro-programme chargé d'exécuter l'instruction.

LM a le format suivant :

```
|____LM____|*|////////////|////|N° Macro|
```

Elle charge à l'entrée de la table correspondante au n° de la Macro, le contenu de (A) 9 : 16.

Les 8 premiers bits de chaque mot de la table sont l'appel d'une micro-instruction de branchement.

III. 3.2.3. LME type b

```
|_0 0_|_EXT_|/////|
|/////|____A______|
|/////|____B______|
|/////|____C______|
```

Cette instruction, semblable à LM charge dans la table des Macros Etendues, à l'adresse = au code étendu, le contenu de (A)9:16.

Son format est défini comme suit :

```
|____LME____|*|//////////////|Code Macro Etendue|
```

III. 3.2.4. Protection des Macros existantes.

Le bit de droite du code opération de LM et LME (marqué "x") précise si il faut tester ou non la valeur de la table avant de la charger.

Si ce bit est à 0 aucun test n'est effectué.
Si il est à 1 et si l'ancien contenu est ≠ 0, aucun chargement n'est effectué et une interruption de programme est déclenchée.

Ce mécanisme permet de protéger une macro déjà chargée.

III. 3.2.5. Gestion des Tables.

Nous laissons à titre d'exercice la définition d'une instruction délivrant une entrée libre dans la table LM ou LME.

III. 3.3. Bilans.

Concrétiser le mécanisme sur l'exemple de la MACRO ADD détaillé en 3.1.1.3. nous donnera quelques lignes de réflexion.
Voyons y une démonstration de faisabilité plutôt qu'une démonstration d'efficacité universelle.

III. 3.3.1. Exemple Implémenté.

Soit la configuration ci-dessous des micro-programmes associés aux instructions L, ST, AD, B, BN.

Lignes micro associées aux instructions.

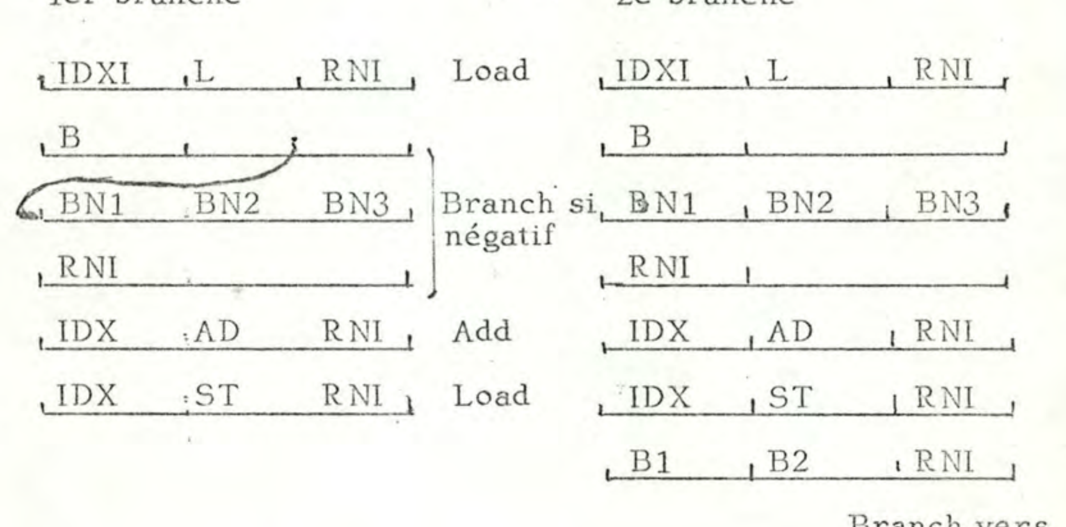
Load	IDXI	L	RNI
Add	IDX	AD	RNI
Store	IDX	ST	RNI
Branch	B1	B2	RNI
Branch si négatif	B		
	BN1	BN2	BN3
	RNI	//////////	

Nous pouvons comparer l'implémentation et le fonctionnement des 3 mécanismes proposés:

- classique
- type a
- type b.

III. 3.3.1.1. Méthode classique.

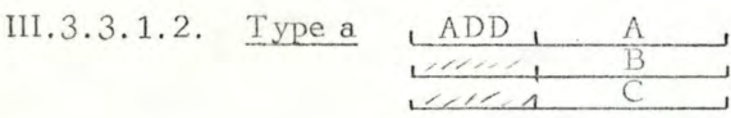
Ligne micro exécutée pour parcourir l'expansion de la MACRO.



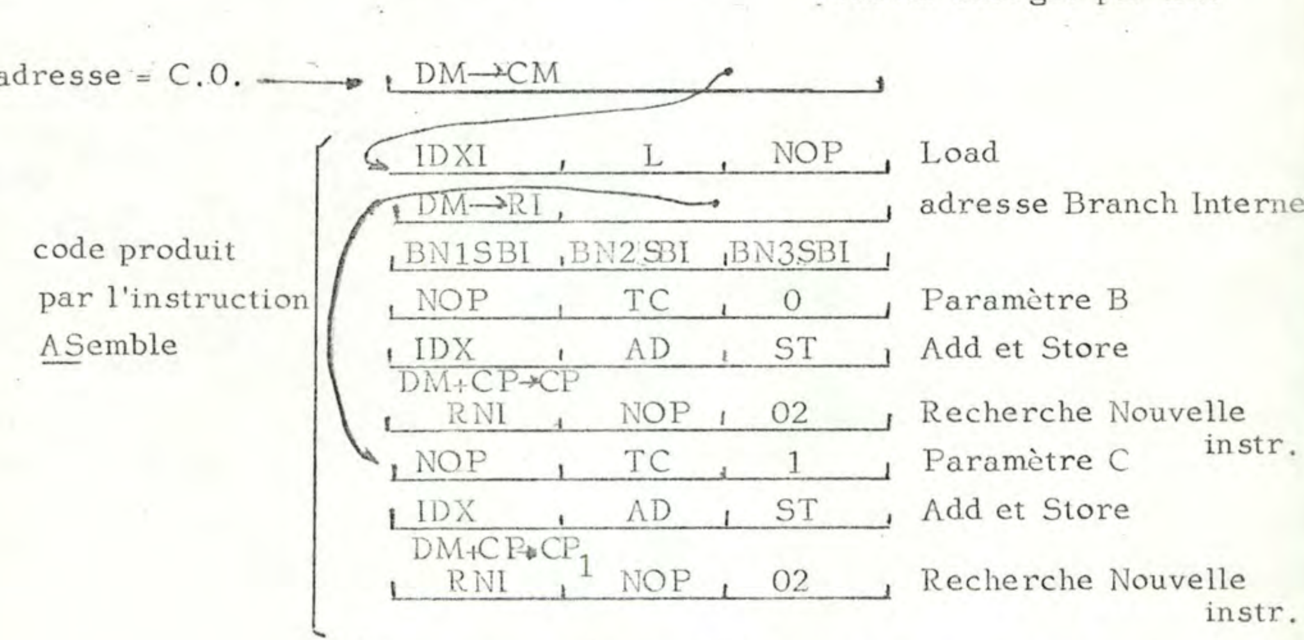
6 lignes micro exécutées

7 lignes micro exécutées.

III.3.3.1.2. Type a



valeur chargée par LM



6 lignes micro exécutées suivant une branche
7 lignes micro exécutées suivant l'autre.

III. 3.3.1.3. Type b

MAC	ADD	//////
//////	A	
//////	B	
//////	C	

Adresse = contenu du point d'entrée = code macro étendu →

	NOP	TC	0	Paramètre A
code produit par l'instruction ASsemble	IDXI	L	NOP	Load
	DM-RI			Adresse Branch Interne
	BN1	BN2	BN3	
	NOP	TC	1	Paramètre B
	IDX	AD	ST	Add et Store
	DM+CP→CP$_1$ RNI	NOP	03	Recherche Nouvelle Instruction
	NOP	TC	2	Paramètre C
	IDX	AD	ST	Add et Store
	DM+CP→CP$_1$ RNI	NOP	03	Recherche Nouvelle Instruction

6 lignes micro exécutées suivant une branche

7 lignes micro exécutées suivant l'autre.

III. 3.2.2. Bilan place mémoire.

	Méthode classique	Type a	Type b
Définition	0	10 mots	11 mots
Application	7 mots	3 mots	4 mots

Si la Macro est appliquée plus de 2, 3 fois, la perte de place due à la définition est comblée.
Si la Macro est employée p lus de 10 fois, la place totale d'occupation sera réduite de moitié pour les types a et b par rapport à la méthode classique.

III. 3.2.3. Bilan temps d'exécution.

Par hasard, le nombre de lignes micro exécutées dans chacune des méthodes est toujours égal à 6 ou 7 suivant la branche choisie. Nous ne pouvons pas en tirer de conclusions.
A partir d'autres exemples, il nous semble cependant que :

- notre mécanisme n'est pas plus coûteux en général.
- si les paramètres sont peu nombreux ou souvent employés dans des instructions consécutives, nous avons un gain de temps non négligeable.

CONCLUSION.

Laissons à d'autres plumes le soin de conclure et de juger l'intérêt d'une recherche sur l'interaction firmware-software.

DIEBOLD FRANCE LE FIRMWARE AOUT 1970.

" Le manque de langages évolués et de processeurs de langages "
" n'est pas dû à des considérations techniques; il s'agit avant "
" tout d'une question de financement. "
" Tant qu'un nombre suffisamment important d'utilisateurs n'aura "
" pas exigé de pouvoir utiliser la micro-programmation pour "
" pouvoir adapter leurs systèmes à des problèmes spécifiques, "
" les constructeurs ne dépenseront pas le temps et l'énergie né- "
" cessaires à la mise au point de ces langages ".

" Il semble que le compilateur FORTRAN de l'IBM 7090 génère un "
" code objet n'utilisant que 19 des 200 et quelques instructions de "
" la 7090. Les 90 % restantes étant perdues ".

A propos de l'efficience d'un choix de micro-programmes fonction d'un problème.

Tiré du manuel de présentation du Multi 8 (nom Européen du Micro 800 Américain).

Ce texte tiré du manuel Anglais est traduit sans modifier le sens mais en ajoutant certaines précisions données dans le contexte :

" L'implémentation d'un compilateur BASIC pour le MICRO 820 "
" nous donnera un bon exemple de l'importance du choix de micro- "
" programmes. "
" Le MICRO 820 a un jeu d'instructions classique et un ensemble "
" de programmes utilitaires conventionnel. "
" Un système BASIC ne supportant qu'une seule conversation a "
" été développé pour ce MICRO 820. Ce compilateur écrit en "
" assembler MICRO 820 occupe approximativement 7500 bytes "
" en Mémoire. Une autre version de l'architecture software- "
" firmware a été développée en doublant la taille des micro-pro- "
" grammes. (passage de 768 à 1536 mots de 16 bits). Ceci a "
" permis une réduction du compilateur de 66 % (passage de 7.500 "
" à 2.500 bytes) ".

Comme résultat nous avons un gain de place mémoire et un temps de compilation sérieusement réduit.

. Version 1.

Taille	Micro 768 mots de 16 bits	=	1536 bytes
	Compilateur	=	7500 bytes
			9036 bytes

Version 2.

Taille	Micro 1536 mots de 16 bits	=	3072 bytes
	Compilateur	=	2500 bytes
			5572 bytes

Différence = 3464 bytes.

Gain relatif = $\frac{3464}{9036}$ ou $\frac{3464}{5572}$ = 37,5 % ou 62 % suivant le point de référence choisi.

Mais depuis longtemps je m'oublie, et "J'ai franchi toute borne "...
Souvenez-vous cependant du proverbe grec : "Souvent un fou
même raisonne bien " ... Vous attendez je le vois une conclusion.
Mais vous êtes bien fous de supposer que je me rappelle mes
propos après cette effusion de verbiage. Voici un vieux mot :
"Je hais le convive qui se souvient "; et voici un mot neuf :
" Je hais l'auditeur qui n'oublie pas."

Donc, adieu ! Applaudissez, prospérez et buvez, illustres initiés
de la Folie !

Eloge de la Folie LXVIII ERASME

BIBLIOGRAPHIE

Signalons tout d'abord l'excellent travail réalisé par le groupe SIGMICRO (ACM). Il nous donne une bibliographie assez complète depuis la création du terme MICRO-PROGRAMMATION jusqu'en 1972.

SIGMICRO NEWSLETTER
Juillet 1972 - Volume 3 - Issue 2
pp. 6 - 56

Nous y ajoutons ou y relevons spécialement :

Architecture of 360/IBM Système 360
Arthur L. Samuel
IBM Journal avril 64
pp. 86 - 101

An Algol-like computer Design Language
Y. CHU
Comm. of the ACM
Vol. n° 8 - n° 10 - oct. 65

New directions in software 1960-1966
par OPLER
Proceedings of the IEEE
Vol. 54 - n° 12 - Déc. 1966
pp. 1757 - 1763

Explicit Parallel Processing Description and Control in Programs for Multi and UNI processor computers
J.A. Garden
AFIPS Conference Proceedings - Vol. 29 - 1966
pp. 651 - 660

Reliability Aspects of the Variable Instruction Computer
Edwin H. Miller
IEEE Transaction on Electronics-computers
Oct 1967. pp. 596 -

Micro Programming Revisited
Proceeding of the ACM - National Meeting
1967

Programming-language oriented instruction streams
par LAWSON H.W., Ir.
IEEE Transaction on Computer
Vol. C-17 - n° 5 - Mai 1968
pp. 476 - 585

The growth of Interest in
Micro Programming
M.V. WILKES
International Summer School on new
Trends in Computer Programming
Copenhagen - Denmark -
12-18 Aout 68

Etude de la Micro-Programmation
Délégation Ministérielle pour l'Armement
DRME Contrat 648/67
Rapport 1 août 68
Rapport 2 juin 69

Référence on Micro-Programming
TIS/R69 MS2
General Electric
7 mars 69

Systems Design of a Dynamic Microprocessor
R.W. Cook and M.J. Flynn
Transaction on Computer C-19
mars 1970

Le firmware
DIEBOLD
Août 1970

Dynamic Microprogramming :
Processor Organization and Programming
A.B. Tucker
Comm. of the ACM
avril 71 pp. 240 - 250

Micro-Programming : The hardware
software Interface
M.V. Wilkes
Proceedings of the 1971 IEEE
Inter Computer Soc. Conference
Boston - sept. 71 -
pp. 93 - 94

Etude d'un assembleur universel paramétrable
DOUSSY
Télémécanique Electrique
CRI Contrat 70090 - juin 72

Design of Microprogrammed Control for
general purpose processor
George HOFF
SIG micro Newsletter ACM
Juillet 1972 - Vol. 3 - Issue 2
pp. 57 - 64

Readings in Micro-programming
P.M. Davier
IBM Systèmes Journal
Vol. 11 - n° 8 - 1972
pp. 16 - 41

Nous mentionnons d'une façon spéciale le travail réalisé à l'INSIMAG (Grenoble) par M. ANCEAU et son équipe dans l'étude et la mise au point du système et du langage CASSANDRE.

Cfr. entr'autres la Thèse de M. Mermet.

APPENDICE - SCHEMAS HARDWARE DE L'EPRON

I. SCHEMA BLOC.

I.1. Panneau avant.

I.2. Scratch Pad.

I.3. Opérateur.

I.4. Mémoire principale.

I.5. A8C.

I.6. Driver-Receiver.

I.7. Banc des Requests.

I.8. Mode transfert - Registre S.

I.9. Mémoire Rapide.

I.10. Entrée.

II - III - IV - OPERATEUR.

V. REGISTRE R8.

VI. REGISTRE D'ADRESSE MEMOIRE RA.

VII. INTERRUPTIONS.

VII. 1. Principe.

VII. 2. Réalisation.

VIII. MODE DE TRANSFERT.

VIII.1. Principe.

VIII.2. Réalisation.

IX. RAMR

N.B. : Les numéros en chiffres romains référencent les pages des croquis; ceux en chiffres arabes les commentaires.

APPENDICE - SCHEMAS HARDWARE DE L'EPRON.

I. SCHEMA BLOC.

I. 1. Panneau avant :

Affichage :

- 6 digits hexadécimaux de données.
- Témoin d'horloge.
- Pas de l'horloge.

Commandes :

- 24 commutateurs de données.
- 1 sélecteur d'affichage.
- Initialisation.
- Reset.
- Figeage - RUN.
- Combinateur de mode de fonctionnement.

I. 2. Scratch Pad :

de mots de 24 bits adressés par le champ 1 pendant la phase de lecture d'un sous temps et par le champ 2 pendant la phase d'écriture.

Elle est réalisée par 6 blocs 7489 de chacun 16 x 4 bits ayant un temps d'accès de 35 ns.

I. 3. Opérateur :

Réalise 16 opérations arithmétiques et logiques sur deux opérandes X et Y chargeables à partir des BUS E.
Le résultat est connecté au multiplexeur de sortie.

5 bits définis à partir du champ 3 transcodé par le schéma de la figure page IV précisent l'opération à réaliser.

Une bascule de Retenue et un bit de Débordement relient logiquement cet opérateur aux autres éléments de l'unité de commande.

Les pages II, III, IV et V précisent la structure et quelques détails.

I. 4. Mémoire-principale :

Par module de 4 K jusqu'à 64 K mots de 24 bits (plus parité). Temps d'accès 350 ns et cycle de 850 ns. (PLESSEY).

Adressée par le registre RA précisé page 6, elle est connectée aux BUS E et au multiplexeur de sortie.
Sa commande de type asynchrone synchronisé se fait par les ordres LL et LE du champ 5.

I. 5. A8C :

Registre de 24 bits formé par la concaténation des 3 sous-registres

RAP : registre adresse périphérique.

R8 : registre de 8 bits de test.

CC : compteur de cycles.

A8C est connecté aux BUS E et au multiplexeur de sortie. Le mode de transfert précise le sous-registre concerné.

RAP est de plus connecté à APR et chargé pendant TOI (interruption). Il sélectionne le STROBE envoyé par les ordres VE - VS et la priorité résettée par l'ordre RP.

R8 est détaillé page V.

Pour CC se référer au tableau de la section I.3.2.6..

I. 6. Driver - Receiver :

Assurent la connection entre les BUS externes et les BUS E (VS) ou entre les Bus externes et le multiplexeur de sortie (VE) sur une"largeur"de 8 bits. (Jusqu'à 24 en "option").

I. 7. Banc des Requests :

Réalise l'encodage prioritaire et masque les requests.
Délivre une adresse à destination de RAP et est connecté au multiplexeur de sortie pour charger RA durant TOI.
Les détails sont donnés page VII.

I. 8. Mode transfert - Registre S :

Assure la connection entre le multiplexeur de Sortie et les BUS E. Cette connection est paramétrable par le champ 6 et assure les modes définis section I. 3.2.2.

La page VIII détaille le circuit.

I. 9. Mémoire Rapide :

Par module, jusqu'à 2K mots de 24 bits, adressée par RAMR (Cfr. page IX) avec un temps de cycle de 100 ns.
- Intel 50 RAM - .

Elle est connectée, d'une part, aux BUS E pour le chargement (Initialisation et dynamisme) et d'autre part, au multiplexeur de sortie et à la mémorisation des 6 champs.

I. 10. Entrée :

La combinaison des $2^{ème}$ et $6^{ème}$ champs définit le registre et le ou les groupes de 4 bits concernés.

II.III.IV. OPERATEUR :

2 registres de 24 bits réalisés par 6 blocs 74100 sont connectés à l'entrée d'un opérateur arithmétique et logique (Chargement par les commandes EX, EY).

Cet opérateur composé de 6 blocs 74181 et un bloc 74182 pour le report accéléré réalise 16 opérations énoncées dans la table de vérité page IV avec un délai de 60 ns. pour les 24 bits.
Le circuit de la page IV permet de transcoder les 4 bits du champ 3 (s. temps 1 et 5) en 5 bits à destination des 74181.
Le résultat (Z) est connecté au multiplexeur de sortie.
La retenue est calculée page III.
Elle est fonction des tests 1XR et DR et des ordres 1R et OR.

Le débordement, le bit 1 de X et celui de Z ainsi que l'égalité à 0 des 24 bits de Z peuvent donner lieu à un saut d'un sous-temps.

La page VIII précise ce circuit de test.

V. Registre R8 :

Ce registre de 8 bits peut être :

- chargé : un bit à la fois (ER8).
- testé : un bit à la fois (R8S).
- remis à 0 : les 4 premiers bits (CLR8).
- connecté au multiplexeur de sortie (SA8C).

Le bit à charger est déterminé par les 3 bits de poids faible du champ 6. La valeur à charger est soit le bit 1 des BUS S, soit la bascule de retenue, suivant la valeur du bit de poids fort du champ 6.

Avec d'autres bits (1 de X, EL et Z = 0) ces 8 bits peuvent être testés pour déclencher un saut du sous-temps suivant, c'est-à-dire faire plus 1 sur le compteur de sous-temps. La commande I inverse la condition de test.

VI. REGISTRE D'ADRESSE MEMOIRE RA.

Ce registre adresse la mémoire principale par ses 16 bits de droite. Ses 8 bits de poids fort sont en scratch pad ou (en option), adressent une mémoire locale rapide.

Les 16 bits de droite sont en fait, un compteur décompteur. Il peut être chargé par l'intermédiaire du BUS E et incrémenté ou décrémenté par les ordres RA + 1 et I, RA + 1.

La sortie de RA est connectée d'une part en multiplexeur de sortie et d'autre part, moyennant un petit circuit, à la sélection d'adresse de la mémoire.

Le petit circuit composé d'une bascule BAMP, d'un OU sur 12 bits et de quelques portes, permet d'ajouter 256 à l'adresse si celle-ci est $\leq$ 15 et si BAMP = 1. (Cfr. section I. 3.2.1.).

VII. INTERRUPTIONS.

VII. 1. Principe :

Transformer une série de signaux extérieurs asynchrones, en une adresse, égale au numéro du signal présent non masqué la plus prioritaire (1).

VII. 2. Réalisation :

En 4 phases de :

- Mémorisation.
- Encodage prioritaire et masque.
- Synchronisation avec le CPU.
- Aiguillage microprogramme.

La mémorisation est effectuée par une bascule du type J.K par request avec une borne de Reset (ordre RP en micro-programme) Une porte permet de donner un signal si un nouveau Request apparaît alors que la bascule n'est pas à 0 (OVERRUN).

L'encodage prioritaire est réalisé en deux couches de logique. La première encode les requests par groupe de 8 pour donner les 3 bits inférieurs de l'adresse. La seconde couche sélectionne le bloc de 8 le plus prioritaire et définit ainsi les 3 bits supérieurs de l'adresse; elle prend par multiplexage les 3 bits inférieurs calculés dans la première couche.

La synchronisation avec le CPU donne une valeur stable entre deux clocks horloge.
Elle est réalisée par deux mémorisationssuccessives suivies d'une comparaison. En cas d'égalité, sauf pendant T0, la valeur est chargée dans APR. Celui-ci est connecté au multiplexeur de sortie et à RAP (adresse de STROBE et de reset).
Un signal (APR $\neq$ 0) défige l'horloge et conditionne T0.

L'aiguillage micro-programme, au niveau de la ligne -micro est conditionné par une bascule forçant le bouclage sur la ligne micro (MCC) et une valeur non nulle d'APR.

---

(1) Un signal porte le nom de Request.
Il informe le CPU d'un changement d'état dans l'interface d'un périphérique et synchronise le micro-programme lors de transmissions de données.

L'équation logique est :

$B = MCC$	$TOB = TO.B.$
$I = \overline{B}.APR \neq 0$	$TOI = TO.I.$
$N = \overline{B}.\overline{I}$	$TON = TO.N.$

TOB ne modifie pas la ligne micro, TOI la charge en fonction de RAP et TON en fonction de CM (compteur micro-programme).

VIII. MODE DE TRANSFERT.

VIII. 1. Principe :

Le décodage du champ 6 est suivi d'un regroupement par diodes puis d'un encodage pour commander les 6 multiplexeurs attachés chacun à une section de 4 bits de E. Ces multiplexeurs sélectionnent la section voulue de S.

VIII. 2. Réalisation :

33 blocs réalisent ce multiplexage et la mémorisation de S :

- 1 décodeur.
- 6 encodeurs.
- 6 bancs de 4 bascules.
- 16 multiplexeurs à 8 entrées.
- 4 multiplexeurs doubles à 4 entrées.

IX. RAMR.

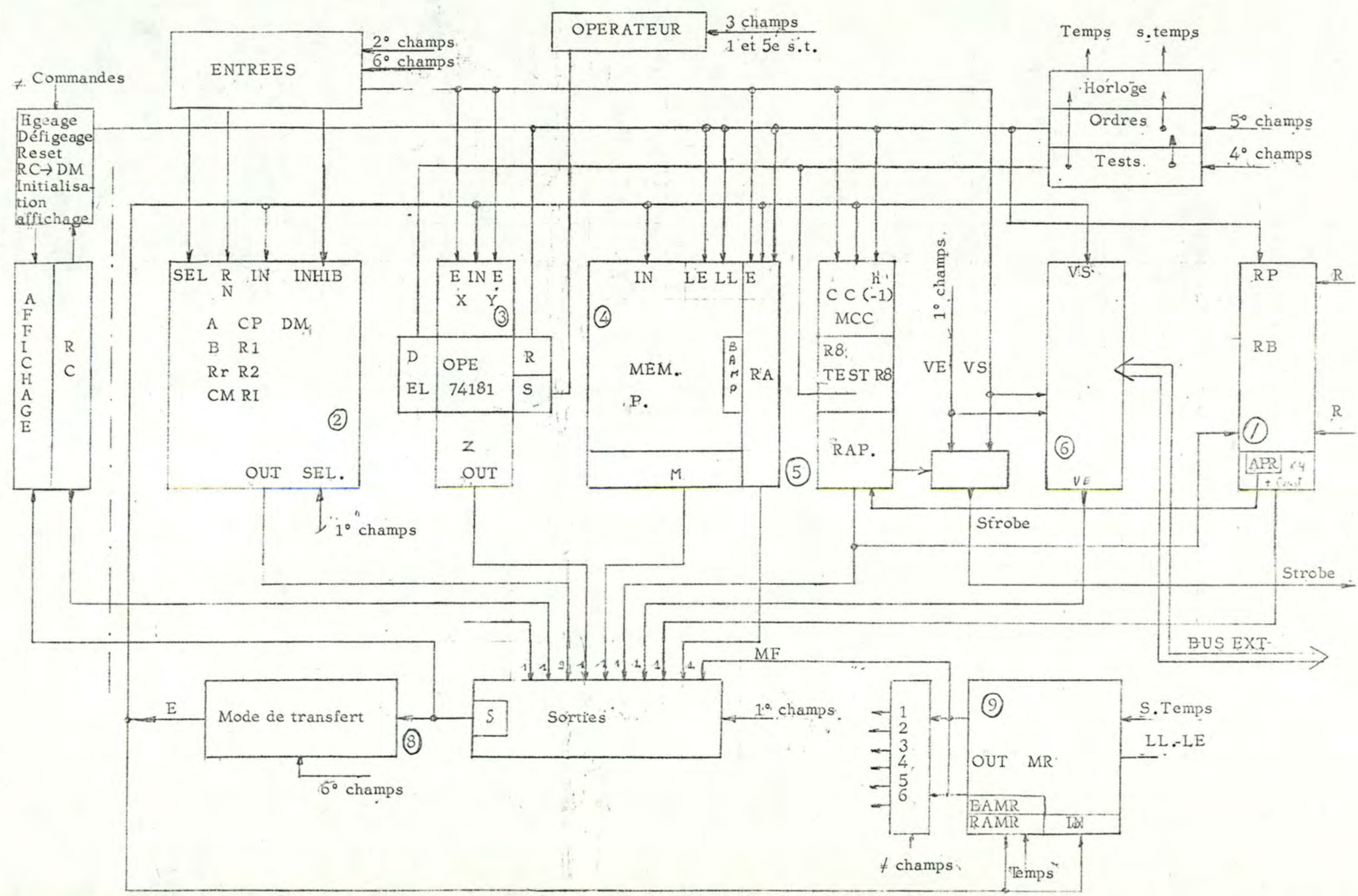
Registre Adresse Mémoire Rapide constitué d'un compteur de 8 bits et d'un sous registre de 3 bits.
Ce registre RAMR de 11 bits permet l'adressage de 2K mots de Mémoire Rapide.

Une bascule (mise à 1 par la détection d'une micro cablée et à 0 par sto) définit le mode de chargement de RAMR, soit à partir de DM via les BUS E pour lire des micro fonctions, soit à partir de A, toujours via les BUS E pour lire ou écrire des données en Mémoire Rapide.
Ce chargement est inhibé après un ordre SEQ et en mode bouclage sur une micro-instruction.

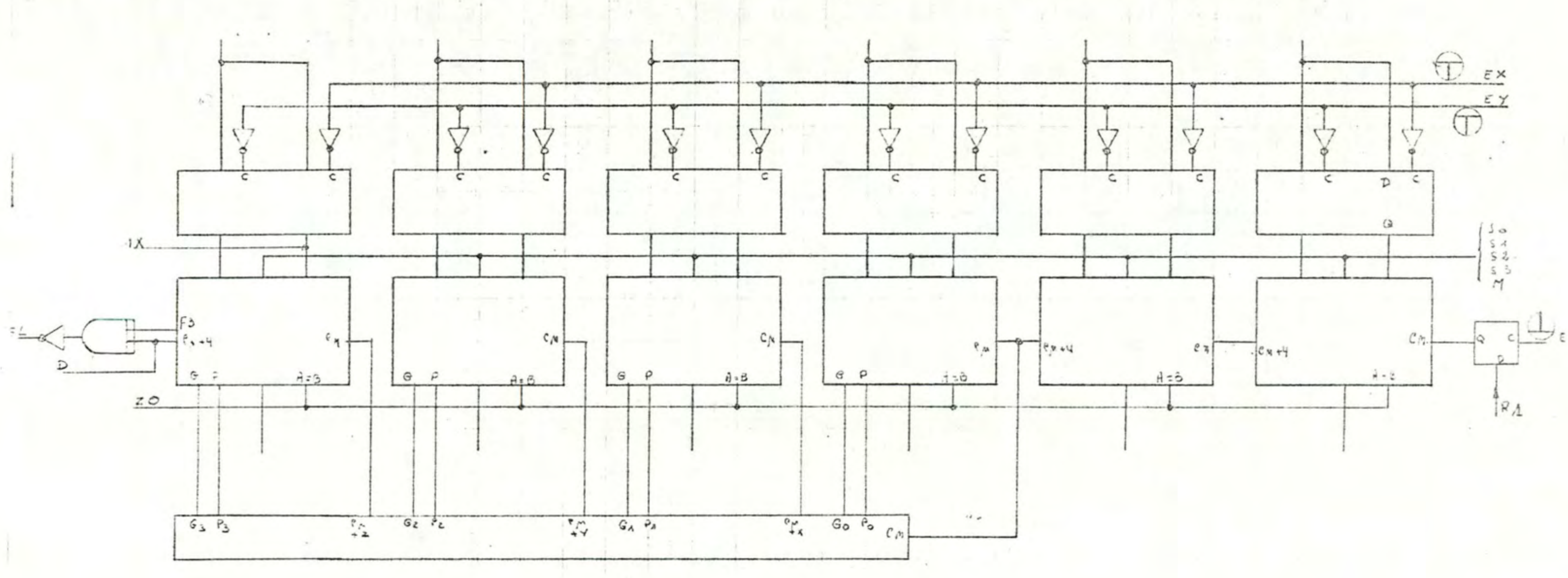
Les ordres SEQ et I, SEQ incrémentent ou décrémentent le compteur de 8 bits. Un multiplexeur à 2 entrées relie le sous registre de 3 bits au compteur de sous-temps.

Une bascule BAMR commande l'addition de 128 à l'adresse si celle-ci est $\leqslant$ 127. (Cfr. BAMP page VI et 7).

EPRON.



Opérateur.

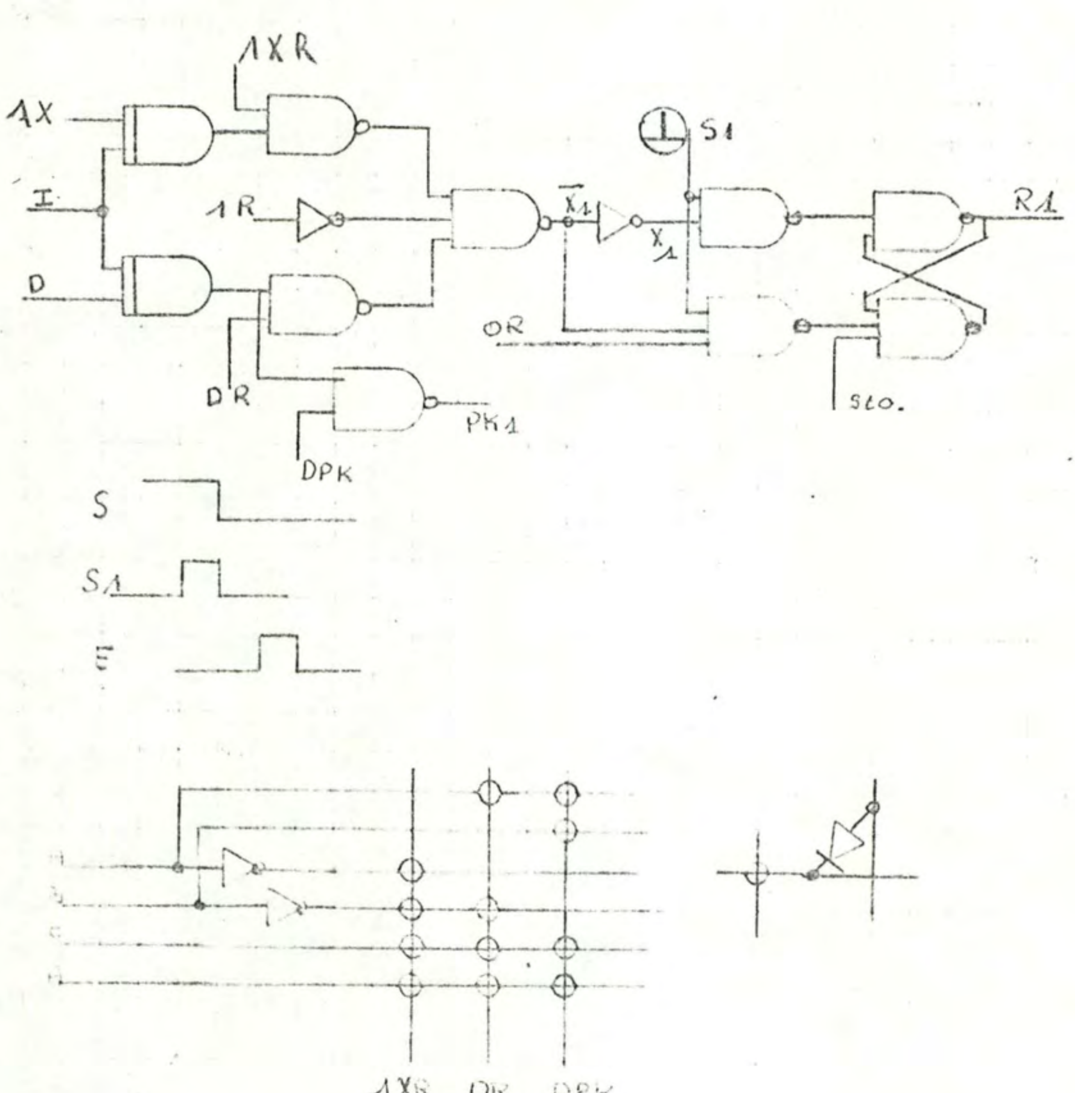


<u>Retenue R 1</u>

$X_1 = 1R + [(1X \oplus I) \circ 1XR] + [(0 \oplus I) . D R]$

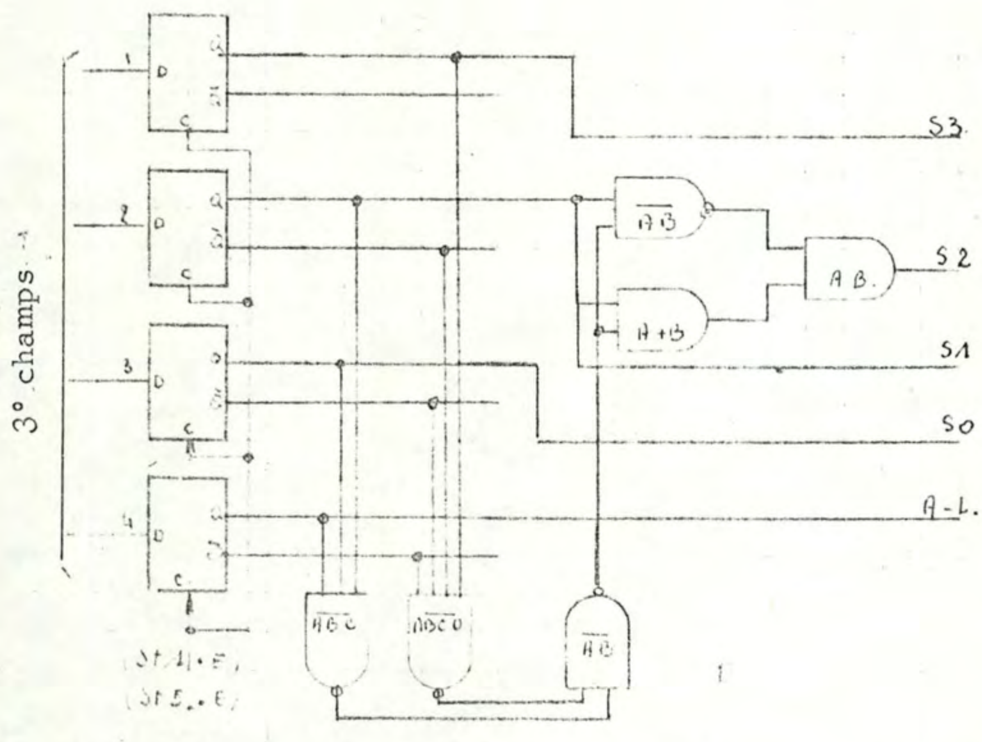
SET $R1 = S_1 . X_1$

RESET $R1 = \bar{S}_1 . \bar{X}_1 . OR$



Commande de l'opérateur

3e champs ⟶ S0 S1 S2 S3 AL(M)



St 1,5

E

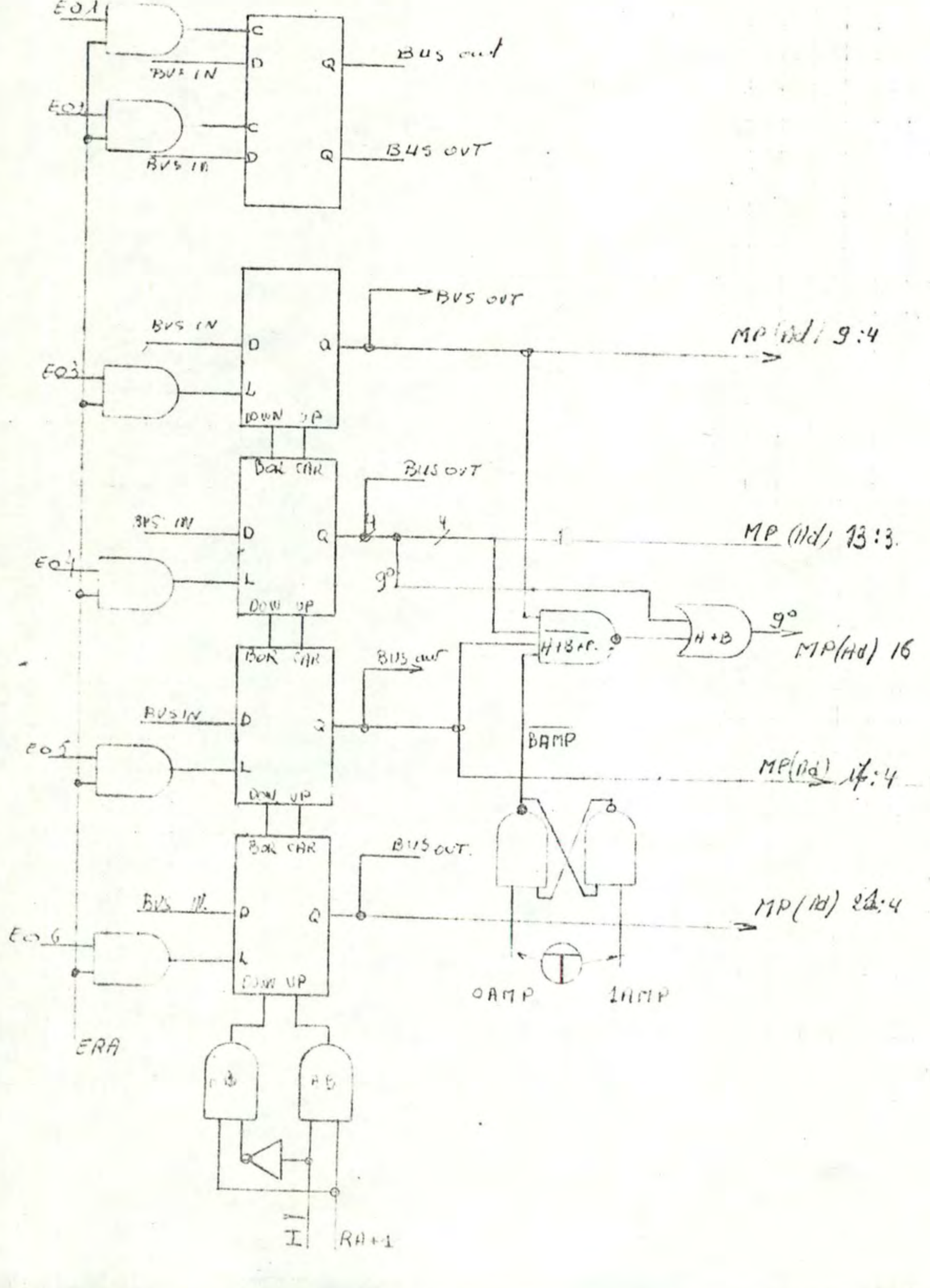
3° champs				Opérateur					
1	2	3	4	S_3	S_2	S_1	S_0	AL	
0	0	0	0	0	0	0	0	0	Z = X + R
0	0	0	1	0	0	0	0	1	Z = $\overline{X}$
0	0	1	0	0	0	0	1	0	Z = (X V Y) + R
0	0	1	1	0	0	0	1	1	Z = $\overline{X \vee Y}$
0	1	0	0	0	1	1	0	0	Z = (X - Y) - $\overline{R}$
0	1	0	1	0	1	1	0	1	Z = X ⊻ Y
0	1	1	0	0	1	1	1	0	Z = $(X \wedge Y)$ - $\overline{R}$
0	1	1	1	0	0	1	1	1	Z = 0
1	0	0	0	1	1	0	0	0	Z = X + X + R
1	0	0	1	1	0	0	0	1	Z = ($\overline{X}$ V Y)
1	0	1	0	1	0	0	1	0	Z = X + Y + R
1	0	1	1	1	0	0	1	1	Z = X ⊻ V
1	1	0	0	1	1	1	0	0	Z = (X V $\overline{Y}$) + X + R
1	1	0	1	1	1	1	0	1	Z = X V Y
1	1	1	0	1	1	1	1	0	Z = X - $\overline{R}$
1	1	1	1	1	0	1	1	1	Z = $X \wedge Y$

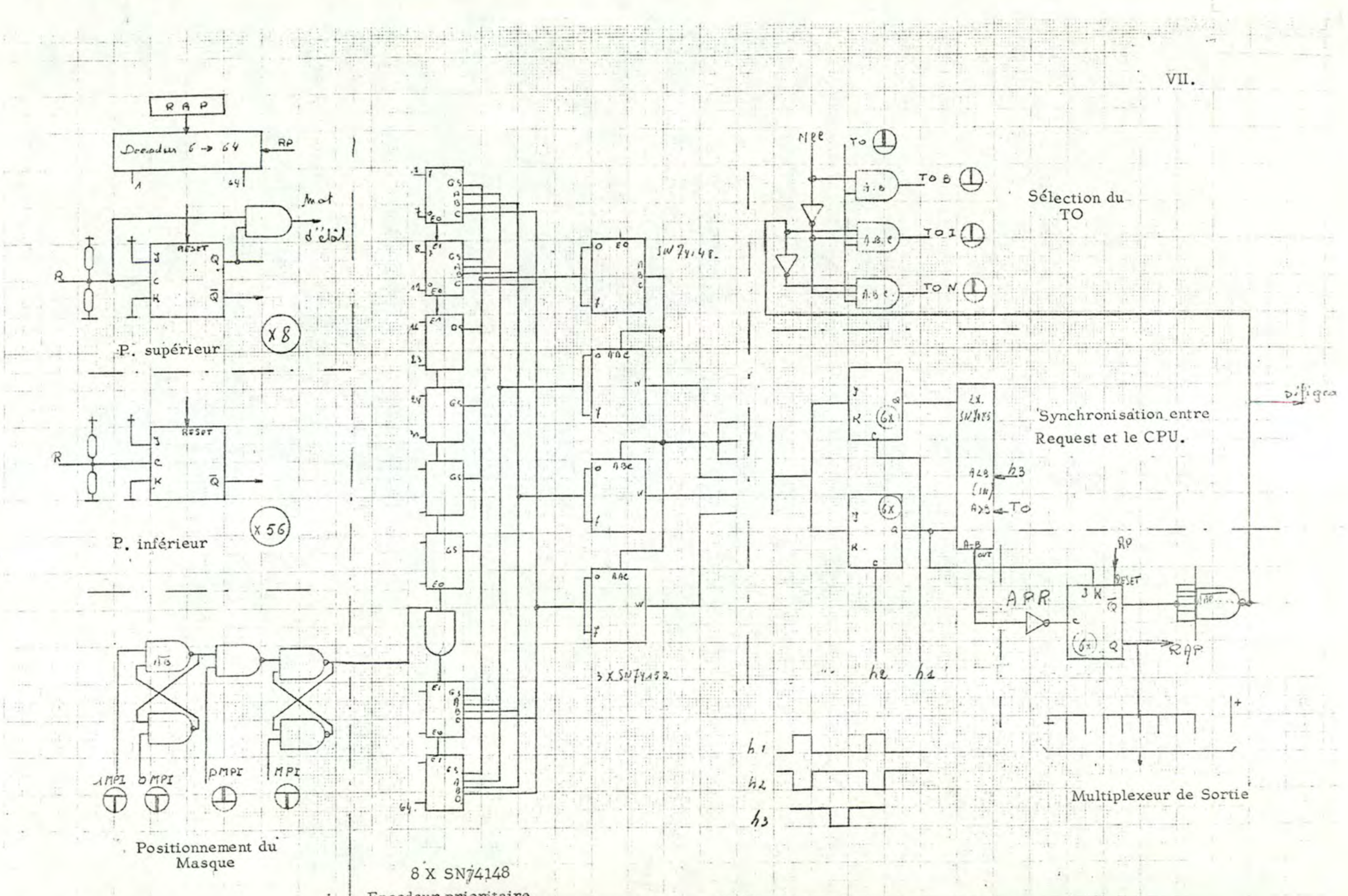
R8

Vers multiplexeur de sortie SA8C

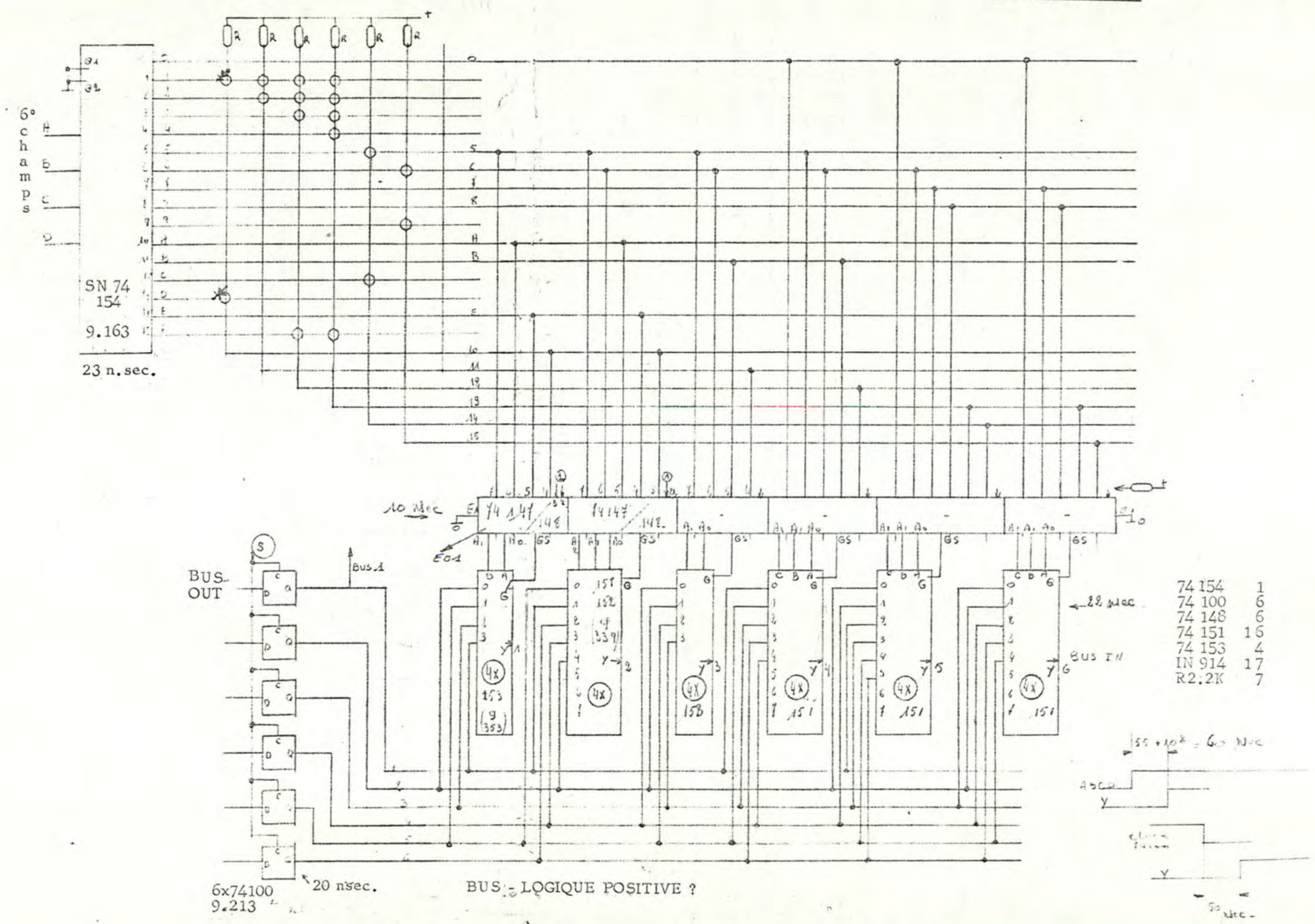
4 7474
1 7442
1 7400
2 7400
2 7404
1 Multiplex.
1 ⊕

RA (MP)




R A P
Decodeur 6 -> 64
RP
Mot d'état
RESET
X 8
P. supérieur
RESET
X 56
P. inférieur
1MPI
0MPI
DMPI
MPI
Positionnement du Masque
SN74148
3 X SN74152
8 X SN74148
Encodeur prioritaire
TO
TO B
TO I
TO N
Sélection du TO
Synchronisation entre Request et le CPU.
h3
TO
APR
RESET
RAP
RP
h2
h1
Multiplexeur de Sortie

Mode de transfert.



RA (MR)

AD M.R.

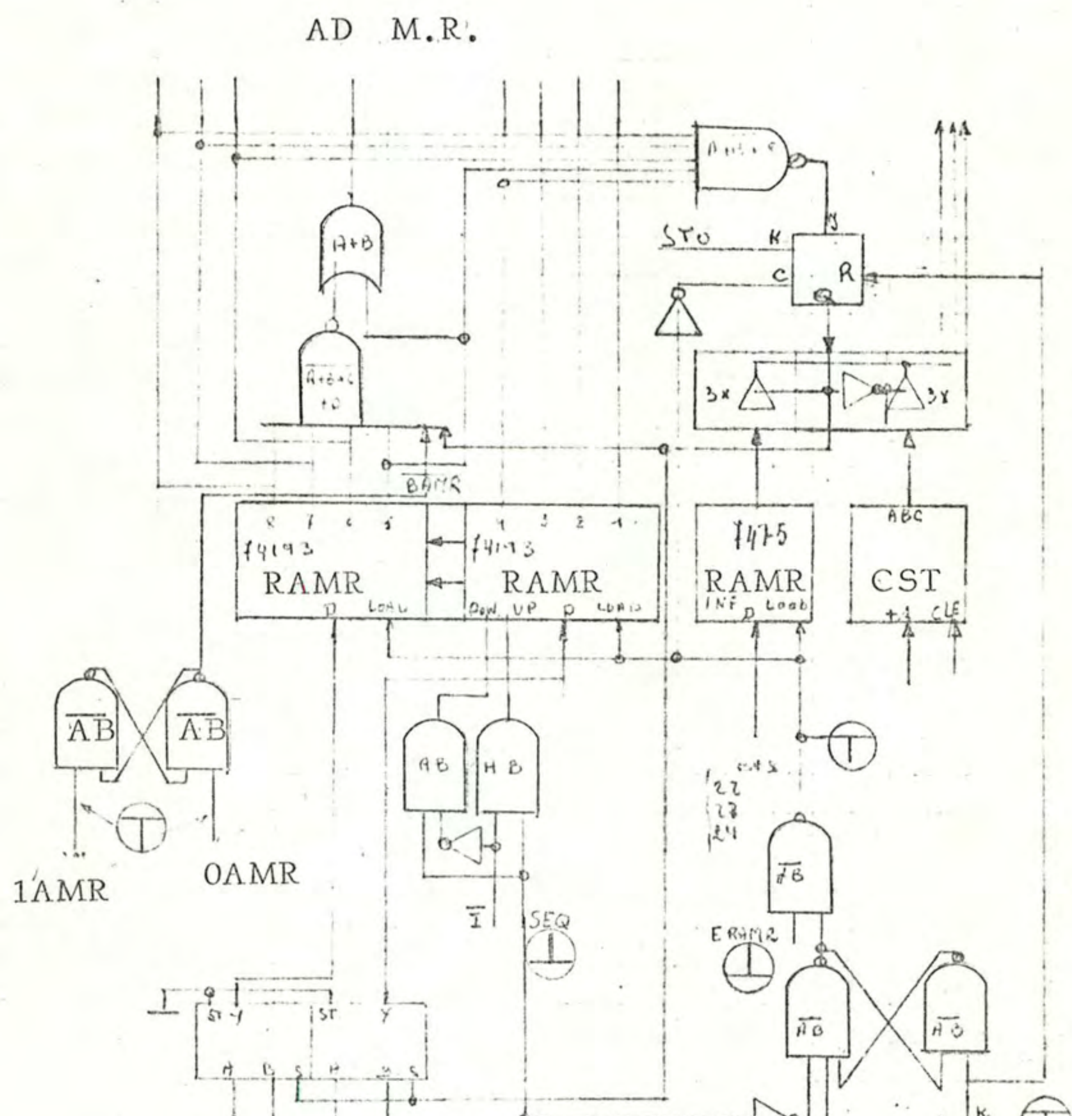


TABLE DES MATIERES

CHAPITRE 0

INTRODUCTION APPROFONDIE.

1. Evolution historique.

2. Prospectives.

3. Les grands axes de la micro-programmation.

   3.1. Contrôle de l'adresse.
   3.2. Décode en champs.
   3.3. Indépendance des mots de contrôle.
   3.4. Lignes de recherches actuelles.
   3.4.1 - Multiplication des niveaux.
   3.4.2 - Dynamisme firmware.
   3.4.3 - Aide à la réalisation de micro-programmes.

4. L'EPRON face à ses axes.

   4.1. Contrôle de l'adresse.
   4.2. Décode en champs.
   4.3. Indépendance des mots de contrôle.

5. Macro assembleur de micro-programmes.

   5.1. Principes.
   5.2. Justification.
   5.3. Réalisation.
   5.4. Points précis d'implémentation.
   5.5. Conclusion et Perspectives.

CHAPITRE I

DEFINITION DU HARDWARE.

1. INTRODUCTION AUX PROBLEMES ET METHODES.

1.1. Méthode de Description d'un ensemble hardware.

1.2. Langage de Description d'un ensemble hardware.

1.2.1. Spécification des registres.
1.2.2. Variables implicites.
1.2.3. Groupe d'instructions - label.
1.2.4. Exécution sélective.
1.2.5. Expression logique.
1.2.6. Constantes.
1.2.7. Variables minuscules.

2. DEFINITION FONCTIONNELLE DE L'EPRON.

2.1. Niveaux et enchaînement.

2.1.1. Première approche essentielle.
2.1.2. Seconde approche plus redondante.

2.2. Eléments de l'EPRON et noms des actions.

2.3. Définition formelle de l'EPRON.

2.3.1. Prise en charge d'une nouvelle micro-fonction.
2.3.1. Principaux "sous-programmes".
2.3.3. Opérateur.

2.3.4.1. Champ 1 - Sorties.
2.3.4.2. Champ 2 - Entrées.
2 3.4.3. Champ 3 - Varia
2.3.4.4. Champ 4 - Tests.
2.3.4.5. Champ 5 - Ordres.
2.3.4.6. Champ 6 - Transfert.

2.3.5. Micro câblées et Boutons poussoirs.

2.4. Description du Simulateur.

2.4.1. Description et Fonctionnement.
2.4.2. Langage de Commande.
2.4.3. Mécanisme spécial d'arrêt.

3. DESCRIPTION INFORMELLE DE L'EPRON.

3..1. Schéma général.

3.2. Composants détaillés.

3.2.1. Mémoire Principale.
3.2.2. Bus et Opérateur de transfert.

3.2.3. Registres.
3.2.4. Opérateur.
3.2.5. Système de décode.
3.2.6. Compteurs et clocks.
3.2.7. Système d'interruption et d'I/O.
3.2.7.1. Description.
3.2.7.2. Fonctionnement.

3.3. Commandes.

3.3.1. Tests.
3.3.2. Ordres.
3.3.3. Entrées-Sorties des registres et bascules.

3.4. Rappel de notation pour la micro-programmation.

3.5. Définition de TO.

3.5.1. TON.
3.5.2. TOI.
3.5.3. TOB.

3.6. Définition de st0 et st8.

CHAPITRE II

REALISATION DE MICRO-PROGRAMMES.

1. ENVIRONNEMENT.
1.1. Format des Instructions.
1.2. Topologie de la Mémoire Centrale.
1.3. Principe recherche nouvelle Instruction.
1.4. Convention d'affectation des registres.
1.5. Adressage.
1.5.1. Indirect.
1.5.2. Indexé.
1.5.3. Direct absolu.
1.5.4. Indexé indirect.

2. INSTRUCTIONS ELEMENTAIRES.
2.1. Load accumulateur.
2.2. Add accumulateur.
2.3. Décrément Mémoire.

3. MICROPROGRAMMES PLUS COMPLEXES.
3.1. Input-Output avec Cadrage et Data Chaining
3.1.1. Organigramme et Micro-Programmation.
3.1.2. Remarques et Commentaires.

3.2. Conversion binaire - BCD.
3.2.1. Organigramme.
3.2.2. Micro-programme.

3.3. Conversion BCD - binaire.
3.3.1. Organigramme.
3.3.2. Micro-programme.

4. PREMIERES CONCLUSIONS.
4.1. Dynamisme au niveau mini-langage.
4.2. Dynamisme au niveau micro-langage.
4.2.1. RNI statistiques.
4.2.2. RNI Trace.
4.2.3. Conditions d'une application optimale.

- - - - - - - -

CHAPITRE III

DEFINITION D'UNE ARCHITECTURE SOFTWARE FIRMWARE DYNAMIQUE.

1. OBJECTIF.
1.1. Hypothèses.
1.2. Propositions.

2. METHODES ET MECANISMES.
2.1. Division en niveaux.
2.1.1. Langage Intermédiaire.
2.1.2. Communication entre niveaux.
2.2. Mécanisme d'appel.
2.2.1. Type a.
2.2.2. Type b.

3. REALISATION.
3.1. Instruction Assemble.
3.1.1. Description du source.
3.1.1.1. Flags.
3.1.1.2. Instructions.
3.1.1.3. Exemple.
3.1.2. Table des micro-programmes.
3.1.3. Mécanisme des branchements internes.
3.1.3.1. Déclaration d'un label.
3.1.3.2. Branchement à un label.
3.1.3.3. Conversion Label adresse.
3.1.4. Organigramme détaillé de l'instruction AS, n
3.1.5. Sous-micro-programmes utilisés.
3.1.5.1. Put μ.
3.1.5.2. GETμ.
3.2. Instruction LM et LME.
3.2.1. RNI modifié.
3.2.2. LM type a.
3.2.3. LME type b.
3.2.4. Protection des Macros existantes.
3.2.5. Gestion des Tables.
3.3. Bilans.
3.3.1. Exemple implémenté.
3.3.1.1. Méthode classique.
3.3.1.2. Type a.
3.3.1.3. Type b
3.3.2. Bilan place mémoire.
3.3.3. Bilan temps d'exécution.

APPENDICE - SCHEMAS HARDWARE DE L'EPRON

I. SCHEMA BLOC.
I.1. Panneau avant.
I.2. Scratch Pad.
I.3. Opérateur.
I.4. Mémoire principale.
I.5. A8C.
I.6. Driver-Receiver.
I.7. Banc des Requests.
I.8. Mode transfert - Registre S.
I.9. Mémoire Rapide.
I.10. Entrée.

II - III - IV - OPERATEUR.
V. REGISTRE R8.
VI. REGISTRE D'ADRESSE MEMOIRE RA.
VII. INTERRUPTIONS.
VII. 1. Principe.
VII. 2. Réalisation.

VIII. MODE DE TRANSFERT.
VIII. 1. Principe.
VIII. 2. Réalisation.

IX. RAMR

N.B. : Les numéros en chiffres romains référencent les pages des croquis; ceux en chiffres arabes les commentaires.