N° d'ordre : 790



# THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

**DOCTEUR DE 3ème CYCLE**

**(INFORMATIQUE)**

par

Pédro CASTILLEJO

# AIDE A LA CONCEPTION DE SYSTEMES A BASE DE MICROPROCESSEURS

Thèse soutenue le 30 Octobre 1979, devant la Commission d'Examen

| | | |
|---|---|---|
| Membres du Jury | C. CARREZ | Président et Rapporteur |
| | J. CORDONNIER | Examinateur |
| | J.M. TOULOTTE | Examinateur |
| | B. TOURSEL | Examinateur |



Ce mémoire est commun aux thèses de P. CASTILLEJO et M. EBOUEYA

d'ordre : 790

# THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

**DOCTEUR DE 3ème CYCLE**

**(INFORMATIQUE)**

par

Pédro CASTILLEJO

# AIDE A LA CONCEPTION DE SYSTEMES A BASE DE MICROPROCESSEURS


Thèse soutenue le 30 Octobre 1979, devant la Commission d'Examen

| Membres du Jury | | |
|---|---|---|
| | C. CARREZ | Président et Rapporteur |
| | J. CORDONNIER | Examinateur |
| | J.M. TOULOTTE | Examinateur |
| | B. TOURSEL | Examinateur |

Ce mémoire est commun aux thèses de P. CASTILLEJO et M. EBOUEYA

NIVERSITE DES SCIENCES
T TECHNIQUES DE LILLE

01.02.1976

## DOYENS HONORAIRES de l'Ancienne Faculté des Sciences

. R. DEFRETIN, H. LEFEBVRE, M. PARREAU.

## PROFESSEURS HONORAIRES des Anciennes Facultés de Droit et Sciences Economiques, des Sciences et des Lettres

ARNOULT, Mme BEAUJEU, MM. BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, CORSIN, DEHEUVELS, HORS, DION, FAUVEL, FLEURY, P. GERMAIN, HEIM DE BALSAC, HOCQUETTE, KAMPE DE FERIET, UGANOFF, LAMOTTE, LASSERRE, LELONG, Mme LELONG, MM. LHOMME, LIEBAERT, MARTINOT-LAGARDE, ZET, MICHEL, NORMANT, PEREZ, ROIG, ROSEAU, ROUBINE, ROUELLE, SAVART, WATERLOT, WIEMAN, MANSKI.

## PRESIDENTS HONORAIRES DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

I. R. DEFRETIN, M. PARREAU.

## PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

. J. LOMBARD.

## PROFESSEURS TITULAIRES

| | |
|---|---|
| BACCHUS Pierre | Astronomie |
| BEAUFILS Jean-Pierre | Chimie Physique |
| BECART Maurice | Physique Atomique et Moléculaire |
| BILLARD Jean | Physique du Solide |
| BIAYS Pierre | Géographie |
| BONNEMAN Pierre | Chimie Appliquée |
| BONNOT Ernest | Biologie Végétale |
| BONTE Antoine | Géologie Appliquée |
| BOUGHON Pierre | Algèbre |
| BOURIQUET Robert | Biologie Végétale |
| CELET Paul | Géologie Générale |
| CONSTANT Eugène | Electronique |
| DECUYPER Marcel | Géométrie |
| DELATTRE Charles | Géologie Générale |
| DELHAYE Michel | Chimie Physique |
| DERCOURT Michel | Géologie Générale |
| DURCHON Maurice | Biologie Expérimentale |
| FAURE Robert | Mécanique |
| FOURET René | Physique du Solide |
| GABILLARD Robert | Electronique |
| GLACET Charles | Chimie Organique |
| GONTIER Gérard | Mécanique |
| GRUSON Laurent | Algèbre |
| GUILLAUME Jean | Microbiologie |
| HEUBEL Joseph | Chimie Minérale |
| LABLACHE-COMBIER Alain | Chimie Organique |
| LANSRAUX Guy | Physique Atomique et Moléculaire |
| LAVEINE Jean-Pierre | Paléontologie |
| LEBRUN André | Electronique |
| LEHMANN Daniel | Géométrie |

| | | |
|---|---|---|
| Mme | LENOBLE Jacqueline | Physique Atomique et Moléculair |
| M. | LINDER Robert | Biologie et Physiologie Végétal |
| M. | LOMBARD Jacques | Sociologie |
| M. | LOUCHEUX Claude | Chimie Physique |
| M. | LUCQUIN Michel | Chimie Physique |
| M. | MAILLET Pierre | Sciences Economiques |
| M. | MONTARIOL Frédéric | Chimie Appliquée |
| M. | MONTREUIL Jean | Biochimie |
| M. | PARREAU Michel | Analyse |
| M. | POUZET Pierre | Analyse Numérique |
| M. | PROUVOST Jean | Minéralogie |
| M. | SALMER Georges | Electronique |
| M. | SCHILTZ René | Physique Atomique et Moléculair |
| Mme | SCHWARTZ Marie-Hélène | Géométrie |
| M. | SEGUIER Guy | Electrotechnique |
| M. | TILLIEU Jacques | Physique Théorique |
| M. | TRIDOT Gabriel | Chimie Appliquée |
| M. | VIDAL Pierre | Automatique |
| M. | VIVIER Emile | Biologie Cellulaire |
| M. | WERTHEIMER Raymond | Physique Atomique et Moléculair |
| M. | ZEYTOUNIAN Radyadour | Mécanique |

PROFESSEURS SANS CHAIRE

| | | |
|---|---|---|
| M. | BELLET Jean | Physique Atomique et Moléculair |
| M. | BODARD Marcel | Biologie Végétale |
| M. | BOILLET Pierre | Physique Atomique et Moléculair |
| M. | BOILLY Bénoni | Biologie Animale |
| M. | BRIDOUX Michel | Chimie Physique |
| M. | CAPURON Alfred | Biologie Animale |
| M. | CORTOIS Jean | Physique Nucléaire et Corpuscul |
| M. | DEBOURSE Jean-Pierre | Gestion des entreprises |
| M. | DEPREZ Gilbert | Physique Théorique |
| M. | DEVRAINNE Pierre | Chimie Minérale |
| M. | GOUDMAND Pierre | Chimie Physique |
| M. | GUILBAULT Pierre | Physiologie Animale |
| M. | LACOSTE Louis | Biologie Végétale |
| Mme | LEHMANN Josiane | Analyse |
| M. | LENTACKER Firmin | Géographie |
| M. | LOUAGE Francis | Electronique |
| Mle | MARQUET Simone | Probabilités |
| M. | MIGEON Michel | Chimie Physique |
| M. | MONTEL Marc | Physique du Solide |
| M. | PANET Marius | Electrotechnique |
| M. | RACZY Ladislas | Electronique |
| M. | ROUSSEAU Jean-Paul | Physiologie Animale |
| M. | SLIWA Henri | Chimie Organique |

MAITRES DE CONFERENCES (et chargés d'Enseignement)

| | | |
|---|---|---|
| M. | ADAM Michel | Sciences Economiques |
| M. | ANTOINE Philippe | Analyse |
| M. | BART André | Biologie Animale |
| M. | BEGUIN Paul | Mécanique |
| M. | BKOUCHE Rudolphe | Algèbre |
| M. | BONNELLE Jean-Pierre | Chimie |
| M. | BONNEMAIN Jean-Louis | Biologie Végétale |
| M. | BOSCQ Denis | Probabilités |
| M. | BREZINSKI Claude | Analyse Numérique |
| M. | BRUYELLE Pierre | Géographie |

.../.

| | |
|---|---|
| M. CARREZ Christian | Informatique |
| M. CORDONNIER Vincent | Informatique |
| M. COQUERY Jean-Marie | Psycho-Physiologie |
| Mle DACHARRY Monique | Géographie |
| M. DEBENEST Jean | Sciences Economiques |
| M. DEBRABANT Pierre | Géologie Appliquée |
| M. DE PARIS Jean-Claude | Mathématiques |
| M. DHAINAUT André | Biologie Animale |
| M. DELAUNAY Jean-Claude | Sciences Economiques |
| M. DERIEUX Jean-Claude | Microbiologie |
| M. DOUKHAN Jean-Claude | Physique du Solide |
| M. DUBOIS Henri | Physique |
| M. DYMENT Arthur | Mécanique |
| M. ESCAIG Bertrand | Physique du Solide |
| Me EVRARD Micheline | Chimie Appliquée |
| M. FONTAINE Jacques-Marie | Electronique |
| M. FOURNET Bernard | Biochimie |
| M. FORELICH Daniel | Chimie Physique |
| M. GAMBLIN André | Géographie |
| M. GOBLOT Rémi | Algèbre |
| M. GOSSELIN Gabriel | Sociologie |
| M. GRANELLE Jean-Jacques | Sciences Economiques |
| M. GUILLAUME Henri | Sciences Economiques |
| M. HECTOR Joseph | Géométrie |
| M. JACOB Gérard | Informatique |
| M. JOURNEL Gérard | Physique Atomique et Moléculaire |
| Mle KOSMAN Yvette | Géométrie |
| M. KREMBEL Jean | Biochimie |
| M. LAURENT François | Automatique |
| Mle LEGRAND Denise | Algèbre |
| Mle LEGRAND Solange | Algèbre |
| M. LEROY Jean-Marie | Chimie Appliquée |
| M. LEROY Yves | Electronique |
| M. LHENAFF René | Géographie |
| M. LOCQUENEUX Robert | Physique Théorique |
| M. LOUCHET Pierre | Sciences de l'Education |
| M. MACKE Bruno | Physique |
| M. MAHIEU Jean-Marie | Physique Atomique et Moléculaire |
| Me N'GUYEN VAN CHI Régine | Géographie |
| M. MAIZIERES Christian | Automatique |
| M. MALAUSSENA Jean-Louis | Sciences Economiques |
| M. MESSELYN Jean | Physique Atomique et Moléculaire |
| M. MONTUELLE Bernard | Biologique Appliquée |
| M. NICOLE Jacques | Chimie Appliquée |
| M. PAQUET Jacques | Géologie Générale |
| M. PARSY Fernand | Mécanique |
| M. PECQUE Marcel | Chimie Physique |
| M. PERROT Pierre | Chimie Appliquée |
| M. PERTUZON Emile | Physiologie Animale |
| M. PONSOLLE Louis | Chimie Physique |
| M. POVY Lucien | Automatique |
| M. RICHARD Alain | Biologie |
| M. ROGALSKI Marc | Analyse |
| M. ROY Jean-Claude | Psycho-Physiologie |
| M. SIMON Michel | Sociologie |
| M. SOMME Jean | Géographie |
| Mle SPIK Geneviève | Biochimie |
| M. STANKIEWICZ François | Sciences Economiques |
| M. STEEN Jean-Pierre | Informatique |

.../...

| | | |
|---|---|---|
| M. | THERY Pierre | Electronique |
| M. | TOULOTTE Jean-Marc | Automatique |
| M. | TREANTON Jean-René | Sociologie |
| M. | VANDORPE Bernard | Chimie Minérale |
| M. | VILLETTE Michel | Mécanique |
| M. | WALLART Francis | Chimie |
| M. | WERNIER Georges | Informatique |
| M. | WATERLOT Michel | Géologie Générale |
| Mme | ZINN-JUSTIN Nicole | Algèbre |

*Nous tenons à exprimer nos remerciements à Monsieur C. CARREZ, Professeur à l'Université de LILLE I qui nous a fait l'honneur de présider le jury de cette thèse. Monsieur CARREZ nous a accueilli dans son équipe de recherche et par ses remarques et conseils nous a aidé à mener à bien ce travail.*

*Nos remerciements vont aussi à Monsieur V. CORDONNIER, Professeur à l'Université de LILLE I et Directeur de l'UER d'IEEA, pour avoir accepté de participer au jury.*

*Nous remercions aussi M. TOULOTTE, Professeur au Laboratoire d'Automatique, pour avoir accpeté de faire partie du jury.*

*Nous exprimons tous nos remerciements à Monsieur B. TOURSEL pour sa collaboration dans l'élaboration d'une partie de ce travail et pour avoir eu l'amabilité de participer au jury.*

*Que Bernard GERMAIN-BONNE et Michel LATTEUX, Maîtres Assistants à l'Université de LILLE I trouvent ici la reconnaissance à leur collaboration.*

*Je voudrais remercier, plus particulièrement, Messieurs DESCARPENTRIES et COMMYN qui m'ont encouragé à venir en France. Avec eux, je voudrais remercier aussi, en espérant qu'ils se reconnaitront, tous mes amis dans le Nord qui m'ont accueilli parmi eux.*

*L'excellente mise en page de cette thèse a été assurée par Bénédicte, Françoise, Michèle et Patricia.*
*Qu'elles soient remerciées pour le soin apporté lors de la frappe de ce mémoire.*

*Je remercie également Monsieur et Madame DEBOCK, pour la rapidité et pour le soin qu'ils ont apporté à la réalisation de cet ouvrage.*

*A Nereida, mon épouse*

*A mes Frères et Parents*

| PAGE | LIGNE | AU LIEU DE | LIRE |
|---|---|---|---|
| 20 | 24 | RANG (VARIAB) de | VECTEUR D'ECHANGE [RANG (VARIAB)] de |
| 24 | 26 | l'application, sans | l'application et des possibilités du matériel µP choisi, sans |
| 25 | 7 | à chaque périphérique_ | à chaque périphérique. |
| 25 | 13 | gestionnaire d'interruption du | gestionnaire du |
| **CHAPITRE 4** | | | |
| 2 | 5 | peut être modifié | peut être modifiée |
| 6 | 1 | en HANDSHAKE | en SHAKEHAND |
| 6 | 4 | exemple ou | exemple où |
| 8 | ——— | ECHANGE DES PAGES 8 ET 9 | |
| 10 | 7 | commande entre | entrée |
| 10 | 13 | décrite de la | décrit de la |
| 14 | 10 | appelons de mot | appelons le mot |
| 16 | 23 | du mot des données COSMIC. | du mot des données. |
| 23 | 3 | dans le chapitre suivant | dans la deuxième partie de la thèse |
| 23 | 6 | réel | réel, ainsi que des algorithmes de choix d'équipement µP. |
| **CHAPITRE 5** | | | |
| 5 | 15 | les circuits d'adressage | les circuits d'accès |
| 10 | 26 | la capacité de poste | la capacité de porte |
| **CHAPITRE 6** | | | |
| 2 | 13 | $C_A = C_p + C_p$ | $C_A = C_P + C_D$ |
| 27 | 16 | Avantages... Inconvénients | Inconvénients... Avantages |
| **CHAPITRE 7** | | | |
| 4 | 15 | du P | du µP |
| 7 | 15 | test de I | test de m |
| 10 | 27 | le P | le µP |
| 13 | | imprimer k(J) | |
| 15 | 3 | (pas 2 ... ... précédente). | |
| **CHAPITRE 8** | | | |
| 10 | 1 | µθ | micro-ordinateur |
| 20 | 15 | pp 119-125 | Journées Génie Logiciel pp 119-125 |
| **CONCLUSION** | | | |
| 3 | 6 | les noyaux d'accès | les moyens d'accès |

**REMARQUE**

LES CHAINES DE CARACTERES SOULIGNEES SONT CELLES QUI SONT OBJET D'UNE MODIFICATION.

# INTRODUCTION

En raison du marché potentiel, en pleine expansion, pour l'utilisation des microprocesseurs, un grand nombre de disciplines et techniques sont utilisées. Ces techniques font appel, entr'autre, à des compétences relevant de l'informatique, au cours des différentes étapes qui caractérisent la construction d'un objet. C'est de cette constatation, qu'est née l'idée de fournir un outil informatique, qui permette à un utilisateur quelconque, de se dégagerdes compétences en informatique et donner libre cours à son imagination.

Notre objectif est donc l'étude et la réalisation d'outils logiciels permettant aux concepteurs de systèmes physiques à base de microprocesseurs, de définir l'assemblage de modules élémentaires et d'obtenir le noyau logiciel d'utilisation du système.

L'outil d'aide à la COnception de Systèmes à base de MICroprocesseur s'appelera COSMIC.

Pour situer COSMIC par rapport aux autres outils d'aide à la conception de systèmes, il nous faut faire un bref rappel sur les modèles de données et de traitement, susceptibles de représenter un système.

La définition la plus intéressante, donnée du système d'information [CAU 79]), nous semble être celle qui consiste a le définir comme image d'un système opérant pour le système de décision. Le système d'information comporte toujours deux éléments :

* les données, statiques ;
* le traitement, dynamique.

La représentation d'un système d'information, se présente donc comme un ensemble de deux modèles : le modèle des données et le modèle traitement, qui ne sont pas toujours indépendants.

Définition d'un modèle

"représentation mémorisée dans l'ordinateur de l'objet en cours de conception et de production. Le modèle pourrait donner une information complète sur l'objet, y compris les spécifications détaillées de fabrication".

## a) Les modèles de données

Leur nombre est relativement élevé. On rencontre, par ordre chronologique d'apparition :

* les modèles de type hiérarchique : la structure des données est une hiérarchie arborescente de plusieurs niveaux

* Les modèles de type réseau : où la structure des données est une combinaison de plusieurs hiérarchies arborescentes.

* Les modèles relationnels où la structure de donnée est définie par un ensemble de relations. Ce type implique une séparation entre le travail de l'utilisateur, (représentation du monde réel : niveau conceptionnel) et celui du concepteur (analyse et définition des structures d'accès : niveau logique).

* Le modèle entité-relations qui se propose de prendre en compte simultanément les avantages des modèles de type réseau et relationnels.

Notre choix a été guidé par :

* un soucis de simplicité d'utilisation

* l'existence d'algorithmes pour obtenir de façon automatique (ou semi-automatique) une partie du niveau logique à partir du niveau conceptionnel.

Pour nous la conception logique commence par l'aide à l'utilisateur, pour qu'il puisse exprimer, grâce au modèle conception (séparation PO-PC, pour COSMIC exclusivement), les structures d'accès aux données tels qu'il les souhaite.
Ainsi COSMIC, dans un domaine restreint, celui de SBμP, s'intéresse aux modèles externes comme Lapage [CAB 79] , et aux contraintes informatiques comme ISDOS et CAM [TEI 71]

b) En ce qui concerne le modèle de traitement, nous avons pris en compte dans COSMIC 3 caractéristiques indispensables qui sont :

- la prise en compte de la notion de temps qui permet une représentation dynamique du système
- la prise en compte de la taille des programmes,
- la prise en compte des opérations non automatisées, qui permet de représenter non seulement l'environnement du SBμP, aspect important pour l'utilisateur, mais aussi d'exprimer les aspects de conception relevant de l'expérience du concepteur et de l'heuristique

Dans le premier chapitre, nous divisons le problème de l'utilisateur en spécifications fonctionnelles et technologiques. Les spécifications fonctionnelles décrivent les différentes fonctions (ou tâches) impliquées dans le problème. Les spécifications technologiques décrivent le fonctionnement et l'utilisation des organes périphériques.

Dans le chapitre 3, les spécifications fonctionnelles sont divisées en deux éléments qui coopérent : partie opérative (ou description des actions) et partie contrôle (ou enchainement et synchronisation des actions). Nous donnons ensuite la structure de ces deux parties.

Le troisième chapitre expose les moyens mis en oeuvre, pour prendre en compte l'échange des variables entre les tâches, et la synchronisation lors de l'exécution de ces tâches. La description du problème de l'utilisateur est achevée, lorsque nous parlons de spécifications technologiques au chapitre 4.

Le chapitre 5 donne un résumé des différents composants d'un système à base de microprocesseurs. Nous dégageons de là, des caractéristiques à retenir concernant le matériel microprocesseur de base.

Le chapitre 6 est consacré à la sélection du matériel microprocesseur de base. Nous étudions plusieurs algorithmes de sélection. Nous en proposons un très simple, après avoir critiqué les algorithmes existants.

Les contraintes de temps d'exécution et de taille mémoire requise sont exposées dans le chapitre 7 . Nous étudions ensuite, quelques algorithmes d'estimation de ces paramètres.

Le chapitre 8, finalement donne l'organisation générale de COSMIC, outil de conception assistée par ordinateur.

M. CASTILLEJO a étudié essentiellement, au long des chapitres 1 à 4, la description du problème de l'utilisateur à destination de l'ordinateur, l'échange d'information entre les tâches et leur mécanisme de synchronisation.

M. EBOUEYA, dans les chapitres 5 à 8, a étudié plus particulièrement, les problèmes liés au choix de l'équipement µP et à l'évaluation des contraintes de temps et taille mémoire qui apparaissent lors de la résolution de problèmes de type temps réel.

# CHAPITRE I : CAHIER DES CHARGES

INTRODUCTION

1.1 ADEQUATION DES µP AU PROBLEME A RESOUDRE

1.2 LES DEUX ENSEMBLES DE CONNAISSANCES

1.3 PASSAGE DU C.C AU C.C.S

1.4 ETAPES DANS LA DESCRIPTION D'UN PROBLEME

1.5 ORGANISATION DU CAHIER DES CHARGES STRUCTURE

1.5.1 Spécifications fonctionnelles
1.5.2 Spécifications technologiques

CONCLUSION

# INTRODUCTION

Le cahier des charges d'un système à base de microprocesseurs, (µP) est un document régissant les rapports entre le fournisseur, *concepteur* d'un matériel de commande et son client, *utilisateur* futur de ce matériel. Un tel document peut faire intervenir des considérations juridiques, commerciales, financières, technico-économiques ou purement techniques [AFC 77]. C'est le premier maillon d'une chaîne aboutissant à la réalisation du Système à Base de microprocesseur [SBµP]

Dans tout ce qui suit, nous nous placerons du seul point de vue du technique. Ce que nous recherchons, c'est avant tout obtenir une description claire, précise, sans ambiguité ni omission du rôle et des performances du SBµP à réaliser.
Souvent cette description est confuse, vague et incomplète, d'où un risque de mauvaise interprétation.

Il faut choisir un point de départ ; le notre a été la nature du problème a résoudre. Il faut ensuite rechercher ou examiner les connaissances permettant d'avancer dans l'étude du projet. Ces connaissances nécessitent souvent une structuration, nous la justifierons après avoir indiqué les étapes possibles de la définition du problème de l'utilisateur.

## 1.1 ADEQUATION DES $\mu P$ AU PROBLEME A RESOUDRE

Une question préliminaire consiste à se demander si le problème a résoudre entre bien dans le domaine des $\mu P$'s. Il est difficile d'y répondre sans avoir consulté une matrice du type de celle que nous présentons à la page 3.

La première colonne de la matrice ne saurait être exhaustive, elle est le point de départ de l'examen.
Notre étude concerne essentiellement les projets de la classe 2, et les frontières de cette classe avec ses deux voisines.

Si par exemple notre projet de SB$\mu P$ est d'avoir une aide à la navigation automatique d'un avion, nous lisons à la 2e colonne que l'exemple courant dans ce genre de projet est de contrôle de processus industriels.
Souvent les objectifs qu'on peut isoler sont :

au niveau du logiciel : diminuer les temps de réaction, développer des stratégies de contrôle.

au niveau du matériel : scruter et vérifier les variables de vol (ex : altitude) par le $\mu P$, et pouvoir communiquer avec le SB$\mu P$ à l'aide d'un périphérique adapté.

C'est un système de classe 2 : un système temps réel.

## 1.2 LES DEUX ENSEMBLES DE CONNAISSANCES

Pour les problèmes susceptibles d'être résolus par un SB$\mu P$, nous prenons en considération les informations suivantes :

1. Celles provenant de la description du problème par l'utilisateur.

2. Celles propres aux microprocesseurs et circuits associés.

| Projet de SB P<br>exemple | Exemple courant | OBJECTIFS SUR LE LOGICIEL | OBJECTIFS SUR LE "MATERIEL" | | | Classe du système |
|---|---|---|---|---|---|---|
| | | | CPU | Mémoire | E/S périphérique | |
| Instrument musical<br>jeux<br>téléphonie | terminal intelligent | introduire des procédés intelligents sur 1 périphérique standard, pour réaliser un produit conforme aux besoins de l'utilisateur | | | minimiser le prix du produit final | interactif homme-machine |
| *contrôle de chaleur et de ventilation<br>*aide à la navigation automatique<br>régulation numérique | processus industriels | développer des stratégies de contrôle pour optimiser la capacité de traitement en utilisant les µP!s.<br>augmenter et obtenir de meilleurs temps de réaction | structation et vérification<br>relation entre l'architecture matérielle et la vitesse | | créer des liens qui permettent la communication entre le SBµP et le processus à contrôler<br>↓<br>système unique | temps réel<br>machine-environnement |
| Calculatrice de poche/mini à base de µP | ordinateur scientifique | choisir les structures de données et des algorithmes performants | | compromis RAMs rapides et mémoires de masse lentes comme les rubans en papier | examiner les différentes imprimantes pour avoir un faible coût | traitement par lot |

Le concepteur ne peut pas toujours être un spécialiste du processus à commander et ne peut pas deviner ce que désire l'utilisateur. C'est le cahier des charges qui servira donc à initier le concepteur. Comme toute initiation, celle-ci doit être progressive si l'on veut que le message "passe" : c'est pourquoi nous proposons la structuration du cahier des charges en le divisant en plusieurs niveaux complémentaires. Le cahier des charges structuré [CCS] ainsi obtenu pourrait aussi être utilisable par un outil informatique.

Les informations propres aux microprocesseurs seront utiles au moment du choix des composants, dont l'ensemble constituera la configuration du $SB\mu^P$ : système adapté au problème de l'utilisateur. Ces informations constituant une base de données où on retrouvera :

a) la description fonctionnelle des circuits appartenant aux différentes familles de $\mu P$'s
par exemple : les fonctions logiques des chips

b) un ensemble de caractéristiques propres à chaque type de chip ou circuit (longueur de mot, mode de fonctionnement, nombre de bits stop, etc).

Ces deux ensembles de connaissances (projet, $\mu^P$'s) permettront à COSMIC de mettre en oeuvre des algorithmes de choix pouvant aboutir à un modèle solution du $SB\mu^P$.
Il restera à faire une vérification de la conformité de cette solution avec le cahier des charges, et à veiller à la cohérence entre la configuration matérielle et le logiciel de mise en oeuvre du $SB\mu^P$.

Nous donnons ci-après un exemple de cahier des charges d'une application que nous utiliserons tout le long de la thèse pour montrer les différentes phases de description du problème de l'utilisateur.

1. La configuration du système de collecte de données [CAP 7E] est composée d'un SBµP, de 3 terminaux, d'une unité de disque et d'une imprimante.

Ce système est utilisé dans une fabrique de lard fumé où les données à recueil sont le poids du porc et d'autres informations telles que : le n° de série, l'identité du cochon, et celle des fournisseurs ... Les données sont saisies aux 3 terminaux. Au niveau de chaque terminal, l'opérateur place l'animal sur une balance reliée au SBµP. Le poids du cochon est alors transféré au SBµp. Ensuite l'opérateur introduit les données supplémentaires.
Le but du système est d'enregistrer ces données sur disquette pour une utilisa tion ultérieure dans un système comptable.
Les données saisies sont imprimées sous forme d'enregistrement homogène pour l'inspection. En résumé, le SBµp doit permettre :

- la saisie de 25 caractères pour chaque animal
- la conversion de ces caractères en un format convenable
- et leur enregistrement sur un secteur du disque (le système comptable impose un cochon par secteur)
- l'édition de ces 25 caractère en une seule ligne avec des blancs comme séparateurs (chaque ligne a donc 50 caractères)
- l'impression d'une en-tête (pour chaque nouvelle page) de 2 lignes de 90 caractères.

2. Le programme doit suivre les opérateurs de telle manière que chacun d'eux puisse saisir 3 cochons par minute. L'intervalle de temps entre la frapp du dernier caractère pour un cochon et la saisie du premier caractère pour le cochon suivant est de 5 secondes.

3. L'imprimante a une vitesse d'impression de 10 caractères par seconde. Il faut 0,15 secondes à l'unité de disquette pour lire ou écrire un secteur de 128 caractères. Les terminaux permettent de saisir 10 caractères par seconde. Nous pouvons ainsi classer les spécifications en deux catégories :

## 1.3 PASSAGE DU CAHIER DES CHARGES AU CAHIER DES CHARGES STRUCTURE

Comme nous l'avons dit, la structuration du cahier des charges nous permettra de réduire notre problème en un ensemble de difficultés plus limitées.
Mais surtout elle nous permettra d'adapter notre cahier des charges à une structure d'accueil : le langage. C'est en effet à l'aide de ce langage de définition que le concepteur introduira le cahier des charges structuré (CCS) dans l'ordinateur.

- spécifications fonctionnelles, qui définissent de façon nette et précise, les différentes fonctions, informations et commandes impliqués dans l'automatisation du processus

- spécifications technologiques, c'est à ce niveau que doivent intervenir des renseignements sur les périphériques de liaison, leur caractéristique et leur mode d'utilisation.

Etapes pour structurer la définition du problème.

1. nous avons les spécifications générales du problème : le CC de l'utilisateur.

2. Ce dernier le transforme pour en faire un C.C.S, écrit en termes du langage de définition. C'est donc à l'utilisateur de faire cette analyse.

3. Le problème de l'utilisateur constitue alors des données fournies à COSMIC.

4. Les données recueillies précédemment ne sont probablement pas complètes. COSMIC demande alors des précisions. Le concepteur y répond donc. Ce peut être des adjonctions, modifications ou même des suppressions de certaines spécifications.

Nous avons choisi pour la troisième étape que les données soient introduites sous forme de fichier-carte, en vue d'un traitement par lot sur ce premier ensemble de connaissances.
La phase 4 est traitée de manière conversationnelle. Pourquoi une partie interactive dans COSMIC ? Parce que la gestion automatisée des différents documents, disponibles ou produits lors de la conception, ne peut être négligée C'est la base de travail. Les différents outils de conception, réalisation et mise au point sont intégrés à cette partie.
De tels systèmes (interactifs) existent déjà dans d'autres domaines d'application ou sont en train d'être étudiés.
Exemple : dans le domaine de l'Architecture avec des organes de dialogues graphique.

Nous avons laisé la phase d'analyse à la seule responsabilité de l'utilisateur. Il n'est pas exclu que le concepteur l'aide à diviser son problème en parties plus petites, chacune exposant une partie limitée du problème. A la fin de cette phase, les informations liées au premier ensemble de connaissances forment un ensemble hiérarchisé d'items.

## 1.4 ETAPES DE LA DESCRIPTION DU PROBLEME

Nous résumons ci-après les premières phases de la démarche classique (et non automatisée) qui aboutit normalement à l'adoption d'une configuration logiciel / matériel répondant aux contraintes du problème posé :

1. formulation du problème à résoudre (définition et objectifs du système)

2. définition des différentes fonctions que doit réaliser le SBuP, pour avoir une idée approximative d'items, donc découpage fonctionel.

3. choix des algorithmes pour chacune des fonctions (car il y a différentes façons de traiter un petit problème)

4. étude approximative des données du $SB\mu^P$ (nombre d'informations à acquérir, leur cadence, leur utilisation)

5. choix des structures de ces données

6. définition des besoins en périphériques

7. examen des grandes lignes du problème d'application (liens de précédences entre les tâches décrites en 2)

8. examen des grandes lignes du logiciel de mise en oeuvre et du programme d'application

Biensûr, ces 8 points ne permettent pas d'aboutir à la solution, nous y reviendrons par la suite. Ils sont néanmoins suffisants pour atteindre l'objectif que nous nous sommes fixés dans cette partie. L'utilisateur remarquera qu'ils sont liés les uns aux autres et qu'il faudra souvent revenir à l'étape précédente, pour mieux définir l'étape actuelle.

C'est le cas, par exemple, lorsqu'on recherche des compromis entre les structures de données et les algorithmes.

Par fonctions à réaliser par le $SB\mu^P$, nous entendons des spécifications (fonctionnelles) qui définissent des actions et leurs enchainement face aux évènements issus du processus à contrôler.

La plus petite entité manipulable lors de la décomposition fonctionnelle est une tâche ; pour être plus précis disons que :

*la tâche* est la plus petite unité de travail qui peut demander une allocation de ressource [CEL 79] (l'unité de travail étant un ensemble de fonctions logiques).

La tâche sera formée :

* d'un certain nombre de variables de communications. Ce peut être avec d'autres tâches ou avec l'environnement (l'extérieur au sens physique du terme et le logiciel de base du SB$\mu^P$)

* d'un programme comportant lui-même un ensemble d'instructions et de variables locales, non accessibles de l'extérieur.

Cette distinction vient du fait qu'il y a deux types de variables manipulées par l'utilisateur :

- celles qui sont utilisées pour échanger des informations entre une tâche et les couches internes du système, ou entre les tâches elles-mêmes, ou entre les tâches et l'environnement.

  Ce sont les variables de communication

- celles qui sont dites variables locales aux tâches.

Dès la phase 4, ce sont des spécifications technologiques et opérationnelles qui sont données par l'utilisateur. Elles apportent des précisions sur le flux de données en entrée et en sortie des périphériques de liaison avec l'environnement, et sur les contraintes à respecter.

Cette approche hiérarchisée du CC, qui rappelle les méthodes de conception descendantes, en facilitera (pour l'utilisateur) la rédaction et (pour le concepteur) la compréhension.
En sériant les problèmes, fonctionnels d'un côté, technologiques de l'autre, elle évite à l'utilisateur de se sentir submergé sous une foule de détails plus nuisibles qu'utiles dans une première approche.

Nous avons parlé de processus à contrôler et d'automatisme : le $\mu^P$ contrôle le processus, et l'ensemble forme l'automatisme. Cette façon de voir un automatisme a été parfaitement formalisée dans des publications récentes en informatique [AFC 77].

## 1.5 ORGANISATION DU CAHIER DES CHARGES STRUCTURE

D'une façon générale, un système (automatisme) peut se décomposer en 2 éléments qui coopèrent : l'un est dit partie contrôle, et l'autre partie opérative.

La partie contrôle prend les variables de contrôles issues de l'extérieur ou des différentes tâches du problème. Elle élabore en sortie

- des ordres et des signaux de synchronisation destinés aux constituants de l'automatisme en vue d'assurer le séquencement des opérations dans la partie opérative.

Cette dernière correspond à la description de chacune des tâches que l'utilisateur a identifiée dans son problème. Elle produit des résultats.

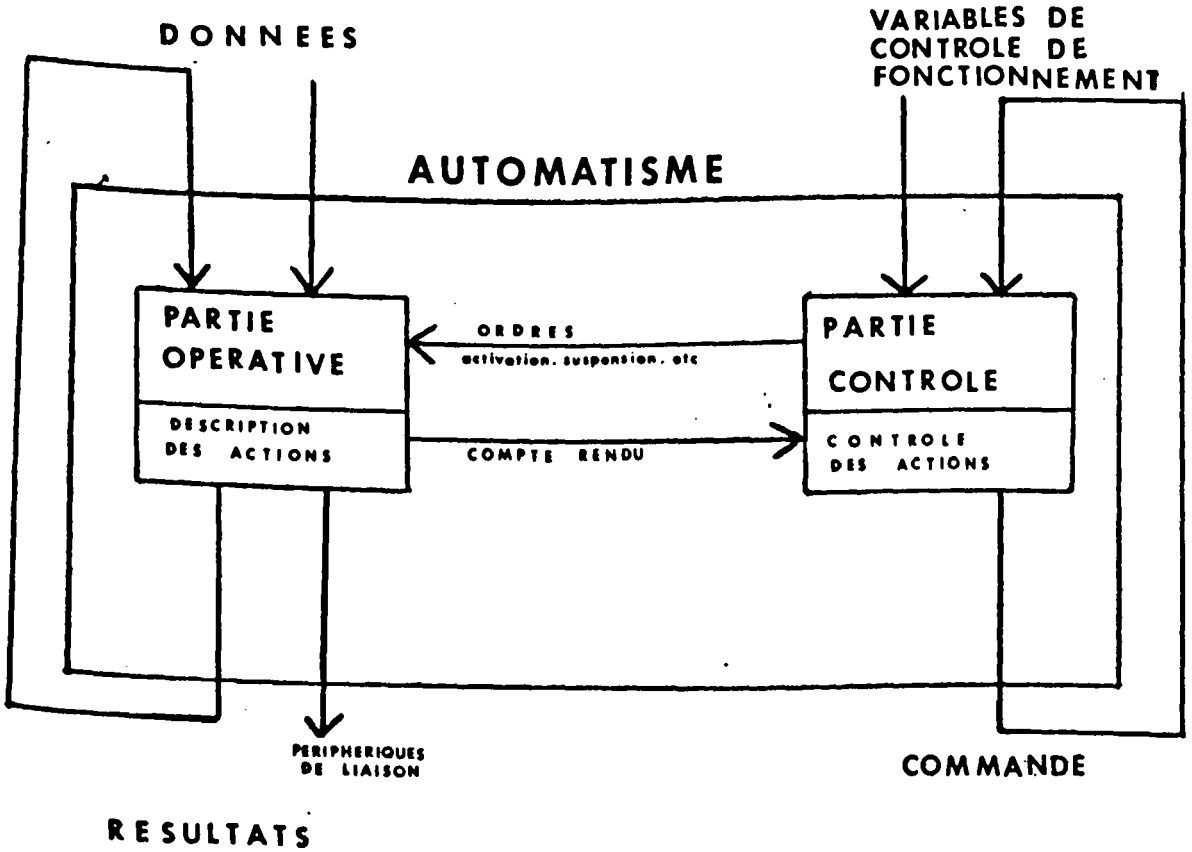



FIG. 2 : SCHEMA SYNOPTIQUE D'UN AUTOMATISME

Dans le schéma précédent, la partie contrôle communique avec la partie opérative par l'intermédiaire d'ORDRES. Ces ordres ont comme rôle l'activation d'une tâche, la suspension d'une tâche, le déclenchement d'une fonction comptage de temps, etc.

Un certain nombre de variables de communication, issues de la partie opérative, sont envoyées à la partie contrôle comme des COMPTES-RENDUS.

Les données produites par la partie opérative, appelées RESULTATS, peuvent être utilisées comme données par d'autre tâches de la partie opérative.

D'autres RESULTATS sont communiqués à l'extérieur par l'intermédiaire des périphériques de liaison.

La partie contrôle effectue des manipulations sur ses propres variabl que nous appellerons variables de SYNCHRONISATION. Le résultat de ces manip tions est utilisé pour le contrôle et l'enchaînement des tâches.

Les contraintes sont :

temporelles : fréquence d'activation d'une tâche,
durée,
etc...

ou liées à la prise en compte des conditions externes et/ou internes pour l'activation (ou non) des tâches.

Toutes les idées développées précédemment doivent aboutir à la structuration du cahier des charges. On a ainsi, pour l'automatisme tout entier :

### 1.5.1 Des spécifications fonctionnelles

a) *Identification de l'automatisme*
où l'on identifie le problème à résoudre, ce qui nous situe dans la figure 2.

b) *Partie opérative de l'automatisme*

description des tâches

et pour chaque tâche

son identification
ses variables de communication
ses variables locales
son algorithme de traitement
sa priorité

c) *Section contrôle de l'automatisme*
C'est ici qu'apparaissent les règles de synchronisation entre les tâches et leur enchaînement. Nous étudierons par la suite, parmi les mécanismes de synchronisation et d'enchaînement, ceux basés sur la séparation entre les parties contrôle (PC) et Opératives (PO).

## 1.5.2 Des spécifications technologies et opérationnelles

Dans la figure 1 nous avons distingué 3 classes de projets de SB $^{p}$ ; les systèmes :

| | | |
|---|---|---|
| intéractifs | : | liaison homme-machine |
| temps réel | : | liaison machine-environnement |
| ceux qui jouent | | le rôle classique d'un ordinateur. |

Ici il s'agit de décrire les unités périphériques qui permettent ces communications. Ce seront par exemple : des téléimprimeurs, des magnétophones à cassettes, des unités à disques souples, des capteurs, des relais, etc.

Elles sont décrites de manière symbolique, et le système d'aide les considère comme des tâches. Nous avons séparé leur description de celle des tâches dans le but de distinguer la description fonctionnelle (de l'application) des considérations technologiques.

Pour chaque unité symbolique nous avons :

a) ses spécifications d'interface qui regroupent l'identification, les types de variables, etc.

b) ses spécifications d'environnement où nous retrouvons :

- les vitesses de réception et de transmission des données,
- les formats de ces dernières,
- les capacités de stockage nécessaires, lorsque l'utilisateur peut les donner.

La description de ces unités symboliques peut se retrouver à deux endroits différents du système d'aide COSMIC :

1. dans le premier ensemble de connaissances comme besoin de l'utilisateur [cf. 1.2]

2. dans le second comme éléments d'un des SBμP existants

Nous montrons ci-après la structuration des informations du cahier des charges donné en exemple et ensuite un schéma général de décomposition de la descripti du problème.

Le traitement peut se décomposer de la manière suivante, entre deux groupes de tâches :

1° Tâche COMMUN

* obtenir un tampon plein
* écrire le tampon sur la disquette
* tester l'écriture
* invertir le tampon en format imprimante
* imprimer

si nécessaire imprimer ENTETE
repeat imprimer un character
until une ligne complète

fin COMMUN

2° Tâche TERMINAL (il y a 3 tâches terminaux)

* obtenir un tampon vide
* attendre terminal prêt
* repeat lire un character

stocker le character dans le tampon
until dernier character

* envoyer le tampon

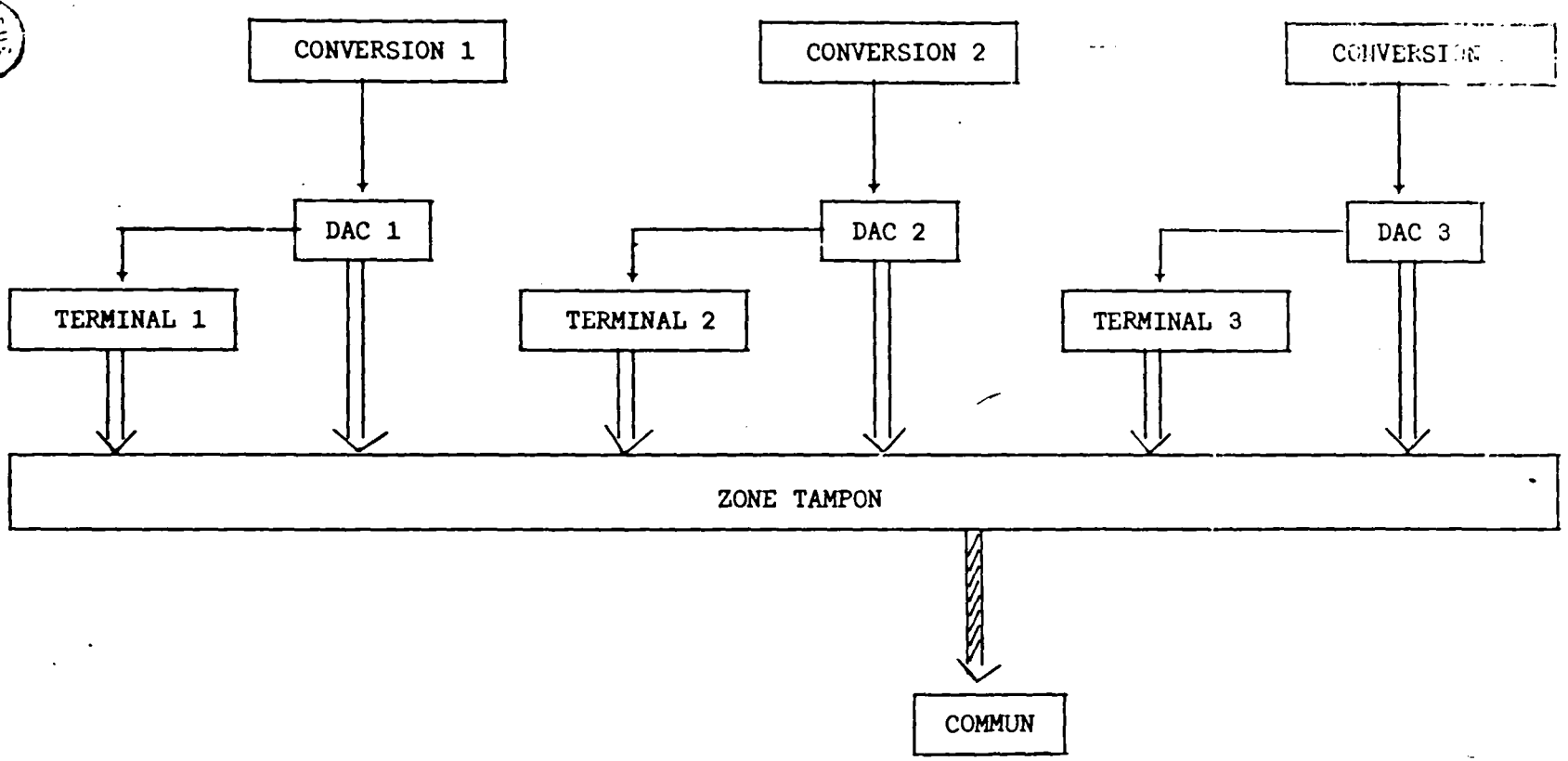
fin TERMINAL

A ces tâches il faut associer les gestionnaires périphériques et la manipulation générale des données. [Cf. Fig. 3]

Ainsi la tâche CONVERSION est la tâche gestionnaire du périphérique CONVERTISSEUR A/D 12 bits. DAC 1 est la tâche que manipule les données issues du convertisseur. Finalement TERMINAL est la tâche que manipule les données issues du clavier.

Il faut remarquer que nous avons négligé la présentation du gestionnaire des périphériques TERMINAL, DISQUETTE et IMPRIMANTE.
Il en sera de même lors de la description des périphériques ou nous nous appuyerons sur un exemple en décrivant le convertisseur analogique digitale 12 bits.

COMMUN, par contre, est la tâche que manipule les données à destination de la disquette et de l'imprimante.




→ signaux de synchronisation

⇒ dépôt de messages

⇛ prélévement des messages

*Fig. 3 : ENCHAINEMENT ENTRE LES TACHES*

## CONCLUSION

Nous avons proposé à l'utilisateur une façon de structurer son cahier des charges ; la démarche est résumée par la figure ci-dessus. [Cf. Fig. 4]

Dans les spécifications fonctionnelles, il donne des informations sur les parties contrôle et opérative de son problème.

Dans les spécifications technologiques il décrit les périphériques, ou du moins en donne les caractéristiques.

Le concepteur ayant suivi les directives données doit envisager maintenant la codification de ses informations en vue d'un traitement automatique. C'est de la description de ces informations que nous parlerons au chapitre suivant.

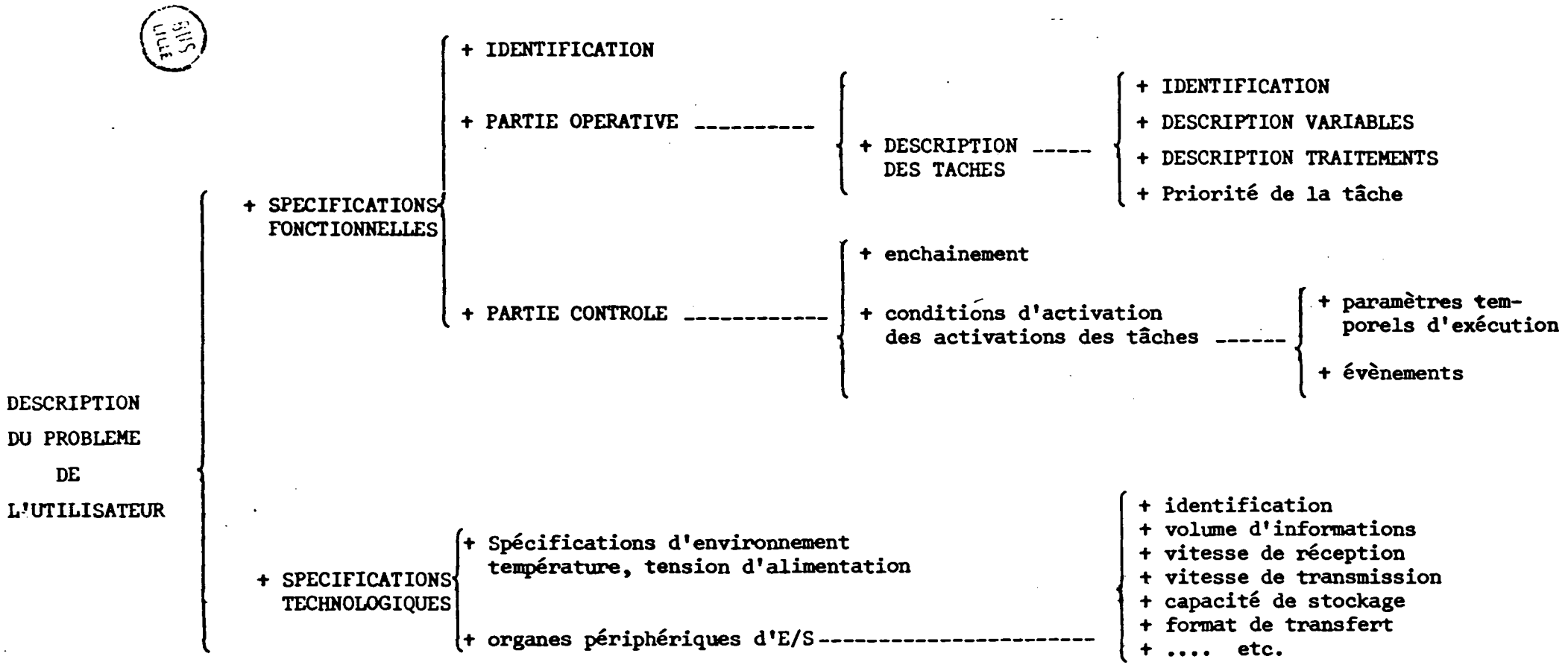



*FIG.4 SCHEMA SUIVI POUR LA PROPOSITION DU C.C.S.*

# BIBLIOGRAPHIE


[AFC 77] Groupe de travail systèmes logiques de l'AFCET.
*Pour une représentation normalisé du cahier des charges d'un automatisme logique.*
Automatique et informatique industrielle, Novembre 1977.

[CAP 78] O. CAPRINI ; S. LAUESEN et U. OUGAARD
*Design Principles for dedicated data collection programs.*
Euromicro 1978, proceeding NORTH-HOLLAND.

[CUS 77] CUSHMAN
*Sharper your microcomputer design skills quickly on microsystems projects.*
EDN, février 1977.

[TEI 71] TEICHROEW
*Méthodes ISDOS : automatisation de la construction de systèmes,*
L'INFORMATIQUE, Déc. 1971.

[WAT 76] WATERS , S.J.
*MAO (CAM) méthodologie assistée par ordinateur dans la conception des systèmes informatiques*
L'informatique nouvelle #69, janvier 1976

# CHAPITRE II : SPÉCIFICATIONS FONCTIONNELLES

INTRODUCTION

## 2.1 PARTIE OPERATIVE

### 2.1.1 Concept de module
### 2.1.2 Structure de la partie opérative

## 2.2 PARTIE CONTROLE

### 2.2.1 Tour d'horizon de quelques mécanismes classiques de synchronisation
#### *2.2.1.1 Module de contrôle*
#### *2.2.1.2 Contrôleur de synchronisation*
#### *2.2.1.3 Expressions de chemin*

### 2.2.2 Quelques critiques aux mécanismes étudiés

### 2.2.3 Structure de la partie contrôle
#### *2.2.3.1 Déclaration des variables en synchronisation*
#### *2.2.3.2 Description des règles de synchronisation*

## 2.3 PRIMITIVES D'ECHANGE ET SYNCHRONISATION ENTRE LES TACHES

CONCLUSION

# INTRODUCTION

La séparation entre les actions à réaliser : PARTIE OPERATIVE et l'enchaînement et synchronisation de ces actions : PARTIE CONTROLE [AFC 77], apporte une grande souplesse lors de l'analyse d'un problème. Etant donné les coûts élevés de mise au point d'une application, cette séparation nous semble une saine politique lors de la phase de développement de l'application. Ceci entraine que la modification d'une tâche est sans effet sur la synchronisation avec les autres tâches et vice-versa.

Nous montrons dans ce chapitre comment peut être définie la partie opérative, Comment le concepteur peut décrire les tâhces à l'aide du langage de définition.

Nous traitons ensuite de la partie contrôle (PC). Notre objectif est d'examiner la synchronisation suivant ses trois couches :

- synchronisation des accès à des valeurs
- Synchronisation avec le temps et les évènements extérieurs
- enchainement de processus (priorité, etc) et ordonnancement.

C'est pourquoi nous parlerons, dans cette deuxième partie :

- des variables de synchronisation
- des règles de synchronisation
- et de certaines primitives qui permettent la prise en compte des variables des tâches pour leurs connexions.

## 2.1 PARTIE OPERATIVE

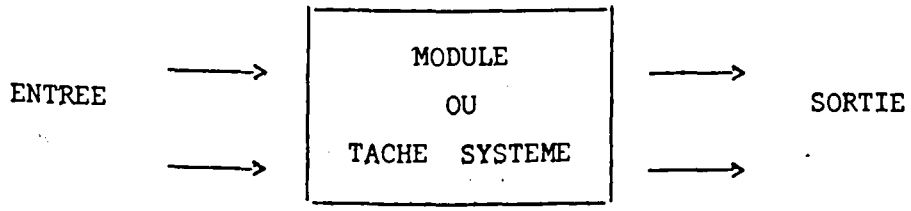
Les tâches décrites dans la partie opérative du problème seront appelées TACHES UTILISATEUR ; l'ensemble des tâches utilisateur avec leur synchronisation constituant ce qu'on appelle PROGRAMMES D'APPLICATION.

Au niveau de COSMIC nous parlerons de TACHES SYSTEME par opposition aux tâches utilisateur.

### 2.2.1 Concept de module

La compexité d'un système réside dans le fait que les différentes fonctions le constituant sont nombreuses et dépendantes les unes des autres. Il est donc important d'organiser clairement ces fonctions. Ceci de manière à faire ressortir des ensembles cohérents, ayant un but commun, des relations simples avec les autres ensembles. Un tel ensemble est appelé module.

Un module est synonime de tâche système. La différence entre tâche utilisateur et tâche système (ou module) réside dans le fait que le concepteur du SBμP ne s'intéresse pas à la façon dont la fonction d'un module est réalisé. Le concepteur considère donc le module comme un "boitier" logiciel avec ses "pattes" d'E/S.




### 2.1.2 Structure de la partie opérative

Nous rappelons que la partie opérative est essentiellement formée de la description de toutes les tâches. Pour chaque tâche le concepteur définit :

- l'identification de la tâche
- les variables
- les traitements sur les variables.

En ce qui concerne les variables, nous les avions jusqu'à maintenant, appelées variables de communication. Il nous semble toutefois qu'il soit nécessaire de distinguer entre :

1. Variables de connexion qui s'utilisent pour la communication de messages entre les tâches.

2. Variables globales[1] qui sont des variables consultables par toutes les autres tâches.

Il nous semble intéressant d'indiquer la place occupée par la partie opérative, lors de la description de l'application par l'intermédiaire du langage de définition. Pour ceci nous décrirons la structure des tâches à l'aide d'un exemple.

Le lecteur se rapportera à l'annexe A pour la syntaxe sémantique du langage de définition.

```
tache   COMMUN
priorité : 5
        global  NBTAMPON : entier,
                TAMPON : tableau [1 .. NBTAMPON]
                        de type tampon,
               TBOOL : tableau [1 .. NBTAMPON]
                       de.booleen ;
        connexion  CHOISI : entier
        VAR ENTETE : chaîne, CHARATER : char
début
(* transférer le tampon sur la disquette *)
écrire  TAMPON [CHOISI] ,
tester.l'écriture ,
attendre. fin. d'écriture. sur. disquette ,
```

---

(1) *ou variables communes.*

```
(* convertir le tampon en format imprimante *)
      convertir  TAMPON [CHOISI] ,
(*libérer le tampon et informer la tâche qui choisit les tampons pleins *)
          TBOOL [CHOISI] = faux ,
           envoyer(TBOOL [CHOISI]) ,
(* imprimer le tampon *)
      si nécessaire imprimer ENTETE,
      faire imprimer CHARATER
      jusqu'à ligne. complète

fin  COMMUN
```

## 2.2 PARTIE CONTROLE

Il existe un certain nombre de mécanismes de synchronisation, qui s'adaptent au principe de base que nous nous sommes fixés : séparation partie contrôle / partie opérative. Parmi eux nous pouvons citer : modules de contrôle, modules de contrôle avec transfert, expressions de chemin, expressions de chemin conditionnel, contrôleur de synchronisation, etc.

### 2.2.1 Tour d'horizon de quelques mécanismes classiques de synchronisation

Le principe de base des mécanismes que nous allons passer en revue, est la distinction entre les actions à synchroniser et la description des règles de synchronisation.

#### 2.2.1.1. *Modules de contrôle*

Les modules de contrôle [ROB 77] prennent en compte deux types d'information :

1. Un ensemble des files d'attente, dans lesquelles seront enregistrées les requêtes.

2. Des règles de synchronisation qui sont exprimées à l'aide de variables servant à compter l'occurrence de certains événements.

L'inconvénient majeur des modules de contrôle, est l'impossibilité d'exprimer la solution de certains types de problèmes. En effet, le mécanisme cité n'utilise que des compteurs d'état qui ne peuvent pas retenir la suite des opérations réalisées, mais seulement leur nombre. Pour y remédier, Robert et Verjus [ROB 78] ont dû considérer deux ensembles de files remplies les unes par des requêtes, et les autres par des opérations de transfert.

La puissance de la méthode provient en grande partie, du fait que l'o permettra de faire dépendre la condition d'autorisation d'une procédure p, de la file dans laquelle elle est placée.

### 2.2.1.2 *Contrôleur de synchronisation*

Le contrôleur de synchronisation [PUL 78], peut être vu comme une généralisation des modules de contrôle. Un des apports étant la liberté laissée au programmeur de faire le choix du nombre, du type, de la structur et de la sémantique de ses variables de synchronisation. Ce qui n'est le cas ni dans les modules de contrôle ni dans les modules de contrôle avec transfert, où les variables de synchronisation sont limitées à cinq compteu Imposer des règles de synchronisation revient à imposer des contraintes dans l'ordre des modifications de la situation du processus vis-à-vis d'une ressource. Ces instants particuliers correspondent à la présence de points de synchronisation (DPS) dans le programme du processus considéré.

### 2.2.1.3 *Expressions de chemin*

Les expressions de chemin [CAM 73], sont utilisées pour exprimer la synchronisation entre processus en décrivant l'ensemble des séquences possibles pour l'exécution des procédures. Chaque expression de chemin est prise en compte par le contrôleur, qui inclut des spécifications. Ces dernières décrivent exactement la façon dont la synchronisation est organis Cette synchronisation s'applique à l'ordre d'exécution des procédures, cell ci pouvant être exécutées par des processus différents. Une expression de chemin décrit donc, un cycle dans l'exécution de l'ensemble des procédures, et accepte comme opérandes des noms de procédures du module.

Les expressions de chemin tels qu'elles sont présentées, ne proposent comme condition d'exécutabilité pour une opération, que l'exécution immédia

antérieure d'une certaine opération. Pour résoudre une plus grande classe des problèmes, aux expressions de chemin ont été ajoutées des notions de conditions d'exécution dans le chemin. L'opérateur ">" a été introduit pour indiquer un choix exclusif avec priorité [FER 78].

### 2.2.2 Quelques critiques aux mécanismes étudiés

Pour des problèmes classiques, il est certain que les mécanismes cités (cf. 2.2.1) apportent une solution satisfaisante lors de la synchronisation des processus. Par contre, lorsqu'il s'agit de synchronisation de processus en temps réel, il n'est pas toujours possible de faire une séparation nette entre partie contrôle et partie opérative.

Ainsi, les deux premiers mécanismes étudiés, ne s'appliquent aux problèmes où les régles de synchronisation font intervenir des paramètres des procédures à synchroniser. Ceci est un handicap certain lorsqu'il s'agit de résoudre les problèmes de synchronisation pour des applications temps réel.

En ce qui concerne les expressions de chemin, chaque description de chemin exprime une contrainte sur l'ordre d'exécution des tâches. Ceci par la spécification des séquences d'exécution admissibles. Ce choix provoque une exclusion mutuelle entre tâches appartenant à un même chemin. Cette exclusion imposée par l'outil lui-même, peut s'opposer à l'efficacité de la collaboration entre processus [FER 78].

Il faut aussi ajouter que malgré cette distinction partie opérative-partie contrôle, dans les expressions de chemin, les actions et le contrôle sur les actions sont finalement réunis dans un seul module [ROB 77].

Nous allons essayer d'apporter une solution adaptée à notre problème. Pour cela nous définissons un mécanisme pour pallier aux problèmes rencontrés lors de l'étude des mécanismes cités. La solution proposée devra, entr'autre rendre possible la communication de paramètres entre tâches et la prise en compte de phénomènes temporels.

Nous devons donc fournir un mécanisme vérifiant les propriétés suivantes :

1. Séparation partie contrôle et partie opérative.

2. Il n'y a pas de restriction quant au choix des variables de synchronisation. Le concepteur peut choisir le nombre, le typ la structure, la signification de leur contenu.

3. L'échange des variables de connexion entre les tâches est à l charge complète du noyau, par l'intermédiaire des primitives d'échange.

### 2.2.3 Structure de la partie contrôle

La description de la synchronisation, à l'intérieur de la partie contrôle, se décompose en deux parties :

1. Déclaration des variables de synchronisation.
2. Description des régles de synchronisation.

#### 2.2.3.1 *Déclaration des variables de synchronisation*

Toute variable, utilisée dans la partie contrôle, est déclarée (implicitement) comme étant une variable de synchronisation. Ces variables peuvent être des variables communes ou des variables de connexion. En particulier, nous aurons besoin de définir des variables de types :

- compteur d'impulsion et temporisateur
- sémaphore
- interruptions et évènement

:

etc.

#### 2.2.3.2 *Description des règles de synchronisation*

Nous nous inspirons pour exprimer des règles de synchronisation des concepts propres à la programmation structure. Ainsi nous distinguons trois constructions de base.

1. Enchainement (ou séquences)
2. Alternative et choix
3. Répétition

L'enchaînement permet de ne pas exprimer une condition d'évolution pour certaines tâches. Ainsi une tâche dont la seule mise à jour des variables de connexion, suffit à la rendre active, n'aura pas besoin d'une condition d'évolution explicite dans la partie contrôle.

L'alternative permet d'exprimer des conditions d'évolution conditionnelles qui tiennent compte de l'état des autres tâches. Nous distinguons deux schémas : SI ... ALORS ... SINON et CAS ... DANS ... . D'autres clauses faisant apparaître le terme APRES induisent aussi une condition. Nous l'utiliserons donc aussi pour exprimer des conditions alternatives d'évolution.

La répétition sert à exprimer des actions périodiques. Ainsi ces régles permettront au concepteur de définir le déclenchement d'une action TOUTES les fois qu'une certaine condition apparait.

Nous pouvons résumer en disant que le concepteur a les moyens d'exprimer deux types principaux de règles : alternative et répétition. La première régle (séquence) est implicitement énoncée dés qu'une tâche n'a besoin d'aucune régle d'évolution. La production de toutes ces variables de connexion suffit dans ce cas à rendre la tâche candidate à s'exécuter.

Nous donnons ci-dessous un exemple de description des régles de synchronisation dans la partie contrôle correspondant au problème défini dans le premier chapitre.

Partie contrôle

Déclaration des variables de synchronisation { I ; NBTAMPON ;TBOOL ; CHOISI ; START, CONVERSION ; STROBE, de la même façon que dans les tâhces. [cf 2.1.2]

début

```
(* obtenir un buffer plein pour COMMUN *)
R1 : pour I = 1  jusqu'à NBTAMPON
     début si BOOL [I] = vrai
           alors CHOISI  = I
           envoyer (CHOISI)
     fin
```

(* démarrage des périphériques CONVERTISSEUR A/D *)

R2 : si START. CONVERSION 1 alors activer CONVERSION 1

R3 : si START. CONVERSION 2 alors activer CONVERSION 2

R4 : si START. CONVERSION 3 alors activer CONVERSION 3

(* démarrage des tâches de manipulation des données issues des convertisse

R5 : si STROBE 1 = vrai alors activer DAC 1
R6 : si STROBE 2 = vrai alors activer DAC 2
R7 : si STROBE 3 = vrai alors activer DAC 3

(* La tâche de surveillance est déclenchée lorsqu'il n'y a pas de travail
pesage et saisie de données *)

R8 : si aucun. porc ou toutes les 24 heures
activer SURVEILLANCE

fin PARTIE CONTROLE

## 2.3 PRIMITIVES D'ECHANGE ET SYNCHRONISATION ENTRE LES TACHES

Un des problèmes rencontrés lors de l'étude des mécanismes de synchro nisation fait au paragraphe 2.2.1, était l'impossibilité de prendre en comp la transmission de certains paramètres des tâches à synchroniser. Nous allo étudier quelques éléments de la façon dont cet échange peut être fait, sans trop nuire à l'indépendance entre la partie opérative et la partie contrôle de l'application.

La primitive ENVOYER, sera utilisée lors de la manipulation des varia de communication dans la partie traitement des tâches, ou dans la partie description des régles de synchronisation. Ceci pour signaler qu'un message est à transmettre. Lors de l'exécution de cette primitive l'exécution de la tâche en cours est suspendue, le temps nécessaire à transmettre le messa aux tâches qui en ont besoin.

La transmission des messages entraine une réévaluation des régles de synchronisation. Il serait donc souhaitable que cette réévaluation n'impliq que les régles contenant le message qui vient d'être transmis.

Pour les variables de connexion, l'utilisation de la primitive ENVOYER, nous parait une solution envisageable pour l'appel du module ECHANGE. Dans tous les cas, la tâche peut être suspendue par une autre têche plus prioritaire qui était en attente du message recemment envoyé.

Pour remédier au problème de susprendre systématiquement l'exécution de la tâche en cours, au profit d'une tâche plus prioritaire, nous adoptons la démarche suivante : nous définissons une autre primitive EXPEDIER. La différence étant que ENVOYER provoque un appel, pour une évaluation immédiate des conditions d'évolution des tâches. Ainsi une tâche qui envoie un message peut se voir susprendre l'accès à l'unité centrale parce que la tâche qui vient de recevoir le message est plus prioritaire. La primitive EXPEDIER, par contre, provoque un appel au module ECHANGE sans évaluation des conditions d'évolution des tâches. Toutefois cette évaluation sera faite lorsque la tâche émettrice rend le contrôle de l'unité centrale.

Les tâches [PARTIE OPERATIVE] communiquent des informations entre elles à l'aide du module d'échange d'informations, lequel communique des comptes-rendus à la partie contrôle qui se charge d'émettre des ordres aux tâches. Les ordres sont communiqués par l'intermédiaire d'un module qui gère les files d'attente des tâches, les compteurs de temps etc .

# CONCLUSION

Nous avons,tout au long du chapitre, étudié la structure des partie opérative et partie contrôle. Ensuite,nous avons proposé une description du problème de l'utilisateur qui tient compte des particularités du problème à résoudre. Finalement nous avons montré avec des exemples les actions entreprises par le module qui manipule ces structures. Dans le chapitre suivant nous étudierons en détail. l'échange de communication entre les tâches et la prise en compte des règles de synchronisation.

# BIBLIOGRAPHIE

[AFC 77] AFCET : Groupe de travail systèmes logiques.
Pour une représentation normalisée du cahier des charges d'un automatisme logique, A2i # 61. Nov. 1977.

[CAM 73] CAMPBELL R.H.
*"The specification of process synchronisation by path expressions"*
Colloque IRIA sur les systèmes opératoires, Feb. 1976.

[FER 78] FERAUD L.
*"Contribution à la définition et à l'implantation du langage d'écriture de systèmes LEST : structures de contrôle"* Thèse 1978, Université Paul Sabatier.

[HAB 72] HABERMANN N.
*"Synchronisation of communicating process"* CACM, march 1972, Vol 15, # 3.

[LAT 78] LATTEUX M.
*"Synchronisation de processus"* Publication # 110, Laboratoire de Calcul de l'USTL, mars 1978.

[PUL 78] PULOU J.
*"Un outil pour la spécification de la synchronisation dans les langages de haut niveau"* RAIRO Vol 12, # 4, 1978.

[ROB 77] ROBERT P. et VERJUS JP.
*"Toward autonomous description of synchronisation modules"* IFIP congress proceeding 1977, Holland.

[ROB 78] ROBERT P. et VERJUS JP.
*"A queue management device for control modules"* A paraitre.

[ROU 78] ROUCAIROL G.
*"Mots de synchronisation"* RAIRO, Vol 12, # 4, 1978.

[THO 79] THOMESSE JP. et DERNIAME
*"Flux de données et synchronisation"* Publication à paraitre.

# CHAPITRE III : NOYAU LOGICIEL

INTRODUCTION

3.1 DEFINITION DU SYSTEME UTILISATEUR

3.2 MODULES COMPOSANTS DU NOYAU

3.2.1 Echange des variables

3.2.2 Exploitation des variables globales et des variables de connexion

3.2.3 Synchronisation des tâches

*3.2.3.1 Choix des règles de synchronisation*

*3.2.3.2 Traitements des conditions*

*3.2.3.3 Traitement des actions élémentaires*

3.2.4 Descripteur de tâche

3.2.5 Module descripteur

CONCLUSION

# INTRODUCTION

La construction d'un système a base de microprocesseur, exige un compromis entre la configuration matérielle de base, et le logiciel que permettra la gestion de ce matériel. Dans ce chapitre nous donnons la définition du système utilisateur que COSMIC doit construire. Ensuite nous étudions le module chargé de l'échange d'information et de l'évaluation des conditions permettant l'évolution des tâches.

Finalement, nous abordons le problème de la prise en compte d'interruptions.

## 3.1 DEFINITION DU SYSTEME UTILISATEUR

Le système que le concepteur va devoir construire, à l'aide de COSMIC, peut se décomposer de la façon suivante [HAN 79].

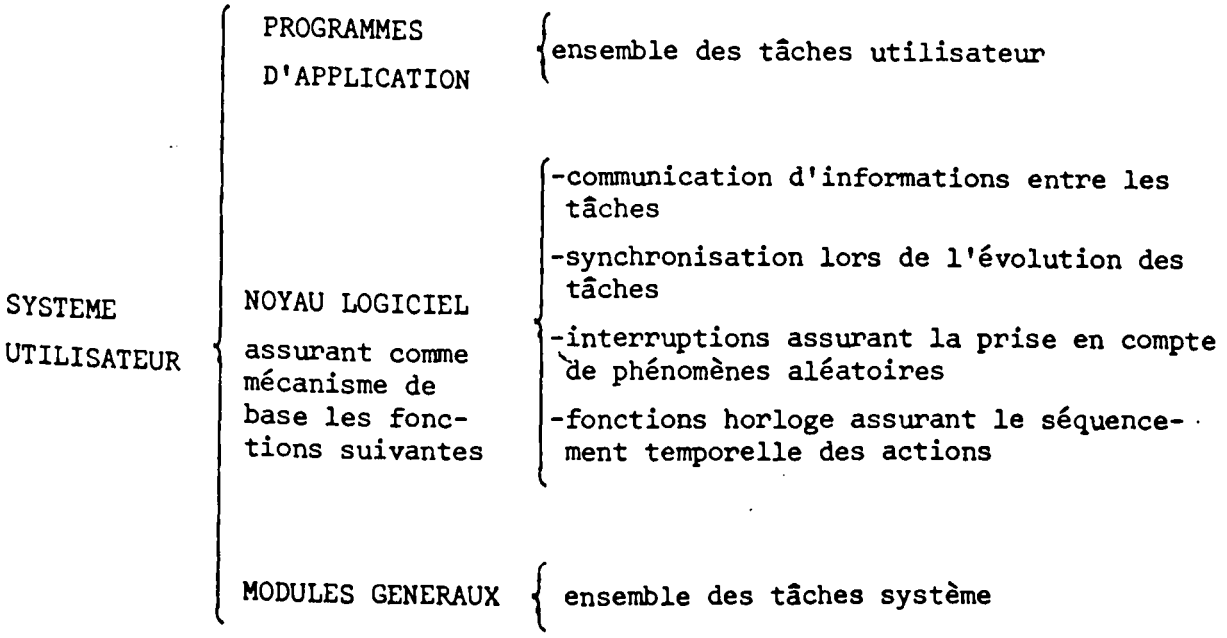



*FIG.1 SYSTEME UTILISATEUR*

Le système utilisateur est un ensemble de couches [FIG 2] dont chacune fournit un certain nombre de fonctions, dépendant seulement des couches internes [LIS 77]. L'interface majeure entre le matériel de base, (système micro-ordinateur) et les programmes d'application, est fourni par le noyau matériel de base qui est la couche la plus interne. Le but du noyau logiciel est de créer un environnement d'accueil pour les tâches. Ceci implique le support des interruptions, le partage du processeur par les tâches et leur mécanisme de communication.

Le noyau doit être adapté à chaque matériel microprocesseur de base, il est la partie du système utilisateur la plus dépendante du matériel. En général, le noyau logiciel est le seul qui a accès aux instructions priviligiés de la couche la plus interne.

Le noyau logiciel est un skeleton aussi général que possible. COSM peut y greffer les structures d'une application quelconque (dans le doma que nous étudions). Il contient les mécanismes de base qui manipulent de objets issus de la description du problème.

La description de l'application, faite en termes du langage de déf tion, est traduite en un code intermédiaire. A cette description s'ajout les modules logiciels définis dans la base des données et dont le concep à besoin. L'ensemble, avec le noyau logiciel, passent par une étape de chargement où les modules seront liés les uns aux autres pour former un appellé système utilisateur.

COSMIC assiste donc le concepteur dans la phase de :

1. assemblage des programmes d'application

2. intégration du noyau logiciel, des programmes d'application et de certains modules généraux pour former le système utilisateur

Le générateur de système permet de construire le noyau logiciel adapté à la configuration matérielle et à l'application décrite [FIG 3].

Nous pouvons schématiser le système utilisateur comme suit :

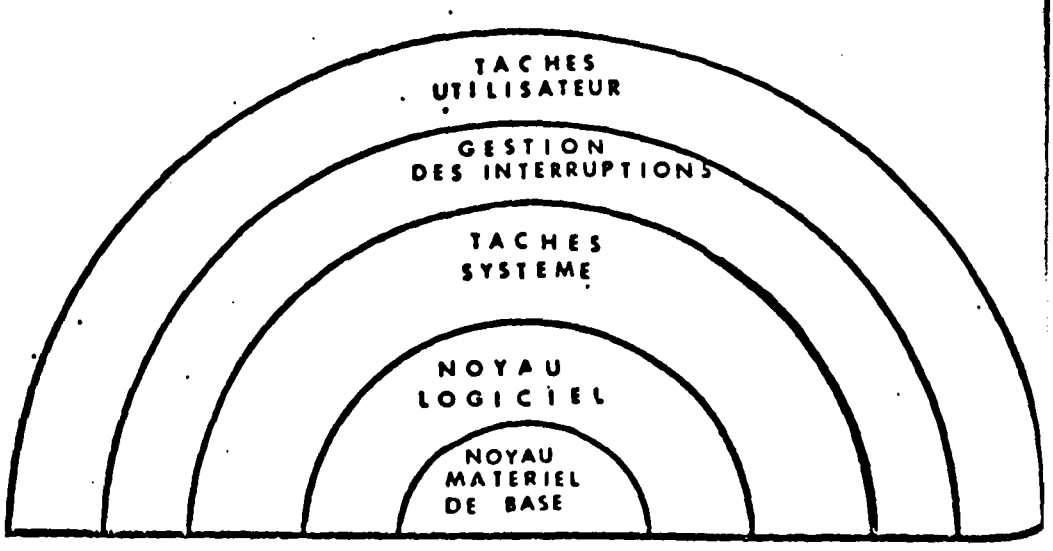



*FIG. 2 : COUCHES DU SYSTEME UTILISATEUR*

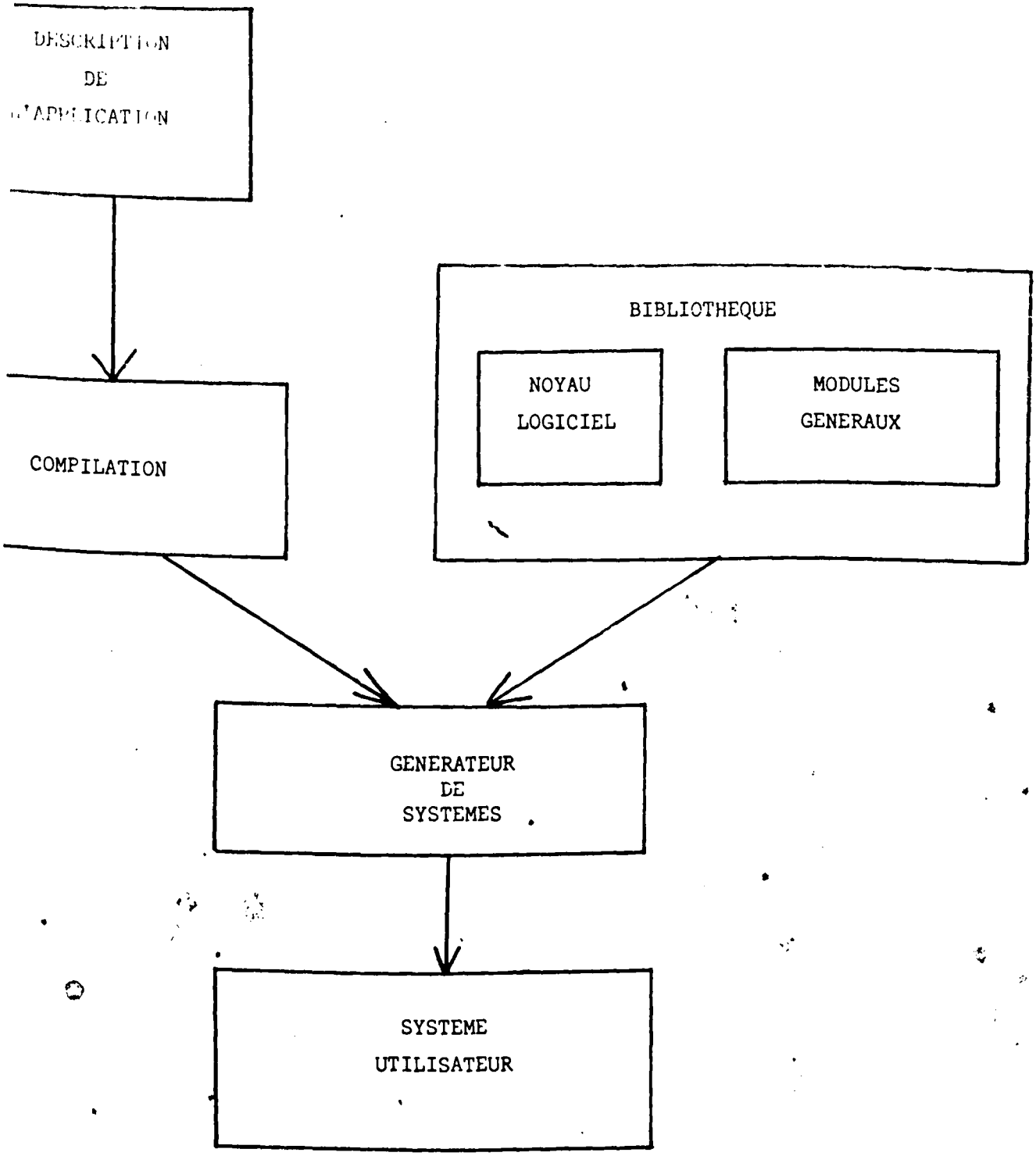



FIG. 3 PHASES DANS LA PROPOSITION DU SYSTEME UTILISATEUR

## 3.2 MODULES COMPOSANTS DU NOYAU

Nous allons décrire ci-dessous les modules logiciel qui font parti du noyau logiciel :

1. module ECHANGE : gère l'échange des variables et la synchronisation entre les tâches

2. module INTERRUPTION

Nous allons décrire séparément, l'échange des variables entre les tâches et la synchronisation des tâches. Ceci malgré leur appartenance à un même module.

### 3.2.1 Echange des variables

Nous étudierons, plus particulièrement, de l'utilisation que nous voulons faire des variables globales et des variables de connexion.

Nous nous inspirons du modèle producteur-consommateur pour la construction de l'échange d'informations. Ainsi, une tâche produit un message (en fait, modifie une variable de communication). Ensuite le module se charge d'avertir les tâches intéressées qu'un message est disponible. Seules les tâches qui ont besoin de ce message en seront averties. Il ne sera par délivré d'accusé de réception par les tâches réceptrices.

REMARQUE : nous rappellons que, un message M est un ensemble de variables A, B, C, ... .

Nous remarquons aussi qu'il n'y a pas une distinction absolue entre tâches productrices et tâches consommatrices. La plupart d'entr' elles peuvent remplir les deux fonctions.

### 3.2.2 *Exploitation des variables globales et des variables de connexion*

Lors de la phase de compilation, COSMIC construit des structures de données, correspondantes aux variables de communications[1] que le conc

(1) variables de communication = (variables globales, variables de conn

teur peut déclarer à l'intérieur d'une tâche. Une de ces structures, que nous n'étudierons pas, est la table de correspondance entre l'identificateur de la variable de communication et l'adresse d'implantation de cette variable.

Il existe une notion de communication attaché aux variables globales et aux variables de connexion. Mais il y a une autre notion plus attaché au concept de synchronisation. En effet, la production d'une variable de connexion A entraine, outre la mise à jour de cette variable, la participation a toutes les tâches qui utilisent la variable A. Ensuite, l'éventuelle activation de la tâche qui n'attends que la variable A. La modification d'une variable globale B n'entraine que la mise à jour de B.

Pour chaque variable globale, COSMIC crée une table avec l'identification de la variable et les droits d'accès à cette variable. C'est-à-dire l'identification des tâches qui utilisent cette variable. Pour qu'une tâche ait accès à une variable globale, il faut que cette variable soit déclarée comme globale dans la tâche.

Les variables de connexion ont besoin, en plus de la table d'identification variable <—> tâche réceptrice, de structures complémentaires.

La première, au niveau des tâches, est due au fait qu'il faut avertir les tâches concernées, de la modification d'une variable de connexion. Cette structure, que nous intégrons à la structure des données correspondante à une tâche, est le VECTEUR D'ECHANGE. Le VECTEUR D'ECHANGE de chaque tâche est un mot avec autant de bits que de variables de connexion déclarés dans la tâche. Ainsi, chaque fois qu'une variable de connexion est modifiée, le module ECHANGE positionne le bit correspondant à cette variable dans le VECTEUR D'ECHANGE de toutes les tâches réceptrices.

La deuxième structure, contient des triplets. Chaque triplet a l'information suivante :

1. Identification de la variable de connexion.

2. Identification de la tâche réceptrice. S'il y a plusieurs tâches réceptrices d'une même variable de connexion, alors COSMIC

construit autant de triplets qu'il faudra.

3. Rang de la variable de connexion lors de sa déclaration dans la tâche réceptrice. Ainsi lors de la déclaration

connexion A,B,C,D ; RANG(C) = 3.

Lorsque une variable VAR, est déclarée dans une tâche TASK1 comme étant variable de connexion, et dans une tâche TASK 2 comme étant une variable globale, la demarche sera la suivante :

1. La variable sera inclue dans le zone des variables globales si toutefois elle n'y était déjà.

2. La variable sera inclue dans la zone des variables de connex C'est-à-dire, que un triplet (VAR, TASK 1, "rang") sera ajo dans la table d'échange.

Ainsi, lors de la modification de la variable VAR, seule la tâche TASK 1 en sera avertie.

En résumé, lors de la compilation des tâches, COSMIC crée la TABLE D'ECHANGE. Celle-ci contient toute l'information concernant les variables de connexion et les tâches réceptrices. A partir de cette table COSMIC construit le VECTEUR D'ECHANGE qui fait partie du descripteur de tâche. La TABLE D'ECHANGE, par contre, est une structure interne au module ECHANGE.

Remarque : auparavant, à chaque déclaration d'une variable globale ou d'une variable de connexion, COSMIC fait l'implantation de la variable dans la ZONE DE VARIABLES GLOBALES ou dans la ZONE DE VARIABLES DE CONNEX respectivement.

Par exemple, lors de la déclaration de la tâche T1, la variable gl P sera ajouté dans la ZONE DES VARIABLES GLOBALES. Les variables de conn seront ajoutées dans la ZONE DES VARIABLES DE CONNEXION. Ceci si ces var n'étaient pas déjà inclues dans l'une ou l'autre de ces tables.

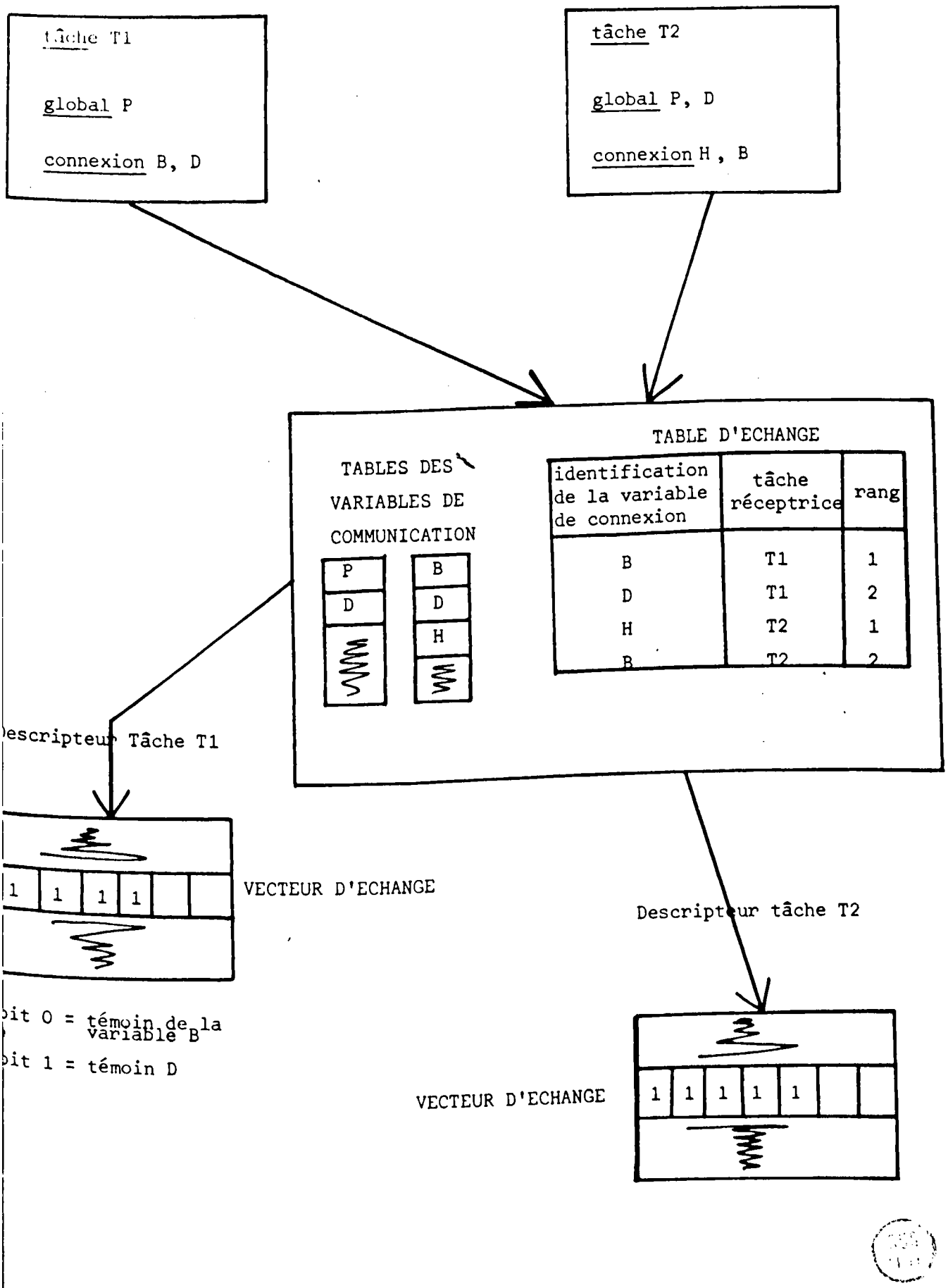



FIG. 4 : SCHEMA DE CONSTRUCTION DES STRUCTURES ASSOCIEES AUX VARIABLES DE COMMUNICATION.

Ensuite COSMIC ajoute, dans la TABLE D'ECHANGE, les triplets corr pondants aux deux variables de connexion B et D.

Finalement, le VECTEUR D'ECHANGE est construit en laissant les bi 0 et/ou 1 pour témoigner de la modification des variables de connexion et D. Tous les autres bits sont mis à 1.

### 3.2.3 *Synchronisation des tâches.*

La partie synchronisation du module est issue de la partie contrô de l'application. Il est constitué des règles de synchronisation défini par l'utilisateur pour contrôler le déroulement de l'application.

Un appel de synchronisation, qui s'effectue lorsque il y a un app au module ECHANGE, provoque les actions suivantes :

1. Choix des règles de synchronisation impliqués.

2. Evaluation des conditions des règles de synchronisation.

3. Actions élémentaires à entreprendre en fonction des règles évaluées.

En particulier, le module ECHANGE rend le contrôle soit à la tâch en cours d'exécution, soit à une nouvelle tâche plus prioritaire que ce en cours d'exécution. Ceci est, biensûr, fonction des primitives d'écha employées [Cf. 2.3].

En général, une règle de synchronisation peut être vue comme comp

1. d'une condition
2. d'une action élémentaire

Ainsi :

a) si <condition> alors <action 1>
sinon <action 2>

CONDITION — ACTION ELEMENTAIRE

b) exécuter TACHE 1 — toutes les 100 millisecondes

ACTION ELEMENTAIRE — CONDITION

sont deux exemples de règles de synchronisation.

### 3.2.3.1 *Choix des règles de synchronisation*

Il nous semble intéressant que, lors de l'arrivée d'un évènement ou message, le module puisse évaluer seulement la (ou les) règle (s) de synchronisation impliquée(s). C'est-à-dire celles où la partie condition contient l'évènement ou les variables récemment modifiées. Pour ceci, nous allons associer, à chaque variable intervenant dans la partie condition, la règle de synchronisation où elle apparait.

Remarque : nous rappellons que les variables de synchronisation sont celles utilisés pour exprimer les règles de synchronisation. En particulier elles peuvent être du type global ou de connexion définis auparavant. Ainsi le concepteur doit déclarer toutes les variables dont il aura besoin pour exprimer la partie contrôle de l'application.

Lors de la compilation de la partie contrôle,COSMIC range les règles de synchronisation dans une zone que nous appellons ENSEMBLE DES REGLES. Au fur et à mesure que les règles sont rangées, COSMIC construit la TABLE D'EVALUATION. Celle-ci est composée de couplets avec pour information :

1. variable qui apparait dans la partie condition de la règle

2. adresse de la règle de synchronisation

Ainsi, pour chaque règle, il y aura autant de couplets que de variables intervenant dans la partie condition de la règle.

COSMIC crée finalement, un mot d'état avec autant de bits que de variables différentes intervenant dans la partie condition des règles. Nous appellons ce mot VECTEUR DE SYNCHRONISATION.

Nous arrivons donc à ce que nous appellons un DESCRIPTEUR DE CONTR
Le descripteur contient tous les renseignements concernant la partie con
de l'application :

1. la TABLE D'EVALUATION

2. le VECTEUR DE SYNCHRONISATION

3. un pointeur vers L'ENSEMBLE DES REGLES

4. un pointeur pour chaque file des tâches : actives, inactive
et en attente.

Lors de la modification d'une variable de synchronisation, le modu
ECHANGE positionne le(s) bit(s) correspond(s) dans le VECTEUR DE SYNCHRO
SATION. Le module consulte ensuite la TABLE D'EVALUATION qui lui donne l'adresse de la règle de synchronisation a évaluer.

### 3.2.3.2 *Traitement des conditions.*

Une condition peut faire intervenir aussi bien le temps que des variables aléatoires[1] ou une expression formée de ces variables.

(1) variables dont la modification n'obéit à aucune loi temporelle.

Ces variables sont appellées en général variables de synchronisation. Elles peuvent être définies comme étant des variables globales ou des variables de connexion.

Selon que la partie condition fasse intervenir des variables à caractère aléatoire, des variables temporelles ou une combinaison, l'évaluation sera faite de la façon suivante :

1. lorsque une variable de connexion est produite, toutes les règles faisant intervenir cette variable dans la partie condition seront évalués. Ceci nous amène à créer une table de correspondance, entre ces variables et les règles de synchronisation où ces variables sont utilisées.

2. Lorsqu'il s'agit de faire intervenir des conditions de comptage de temps, COSMIC créera des horloges logicielles(1) pour chacune de ces conditions. Ainsi la règle de synchronisation sera transformée par COSMIC pour une évaluation plus aisée. Supposons, par exemple, la règle suivante :

   <action élémentaire> toutes les N millisecondes ;

COSMIC va créer une horloge logicielle H1 qui va compter les N millisecondes. A l'horloge logicielle H1, COSMIC associe un évènement EVENT 1, qui sera déclenché lorsque le comptage sera fait. La règle sera donc transformée et mémorisée de la façon suivante :

   si EVENT 1 alors <action élémentaire>

---

(1) Une horloge logicielle est crée à partir des impulsions de l'horloge réel de deux façons : a) une routine de comptage des implusions
b) à partir d'un circuit -temporisateur- compteur d'impulsions, solution plus amplement employé dans le domaine temps réel.

3. l'utilisation des règles du type

<action élémentaire> toutes les R exec. de TACHE 1

amène COSMIC a compter des événements. Pour ceci, il sera réservé un bit dans le mot d'état de la tâche TACHE 1, pour signifier qu'il faut compter le nombre R d'exécutions de TACHE 1.

Au compteur d'évenements sera associé, à la compilation, le signal EVENT 2. La règle de synchronisation sera transformée et mémorisée de la façon suivante :

si EVENT 2 alors action élémentaire

Ainsi, à chaque exécution de TACHE 1, le compteur d'événements associé à cette règle de synchronisation, sera mis à jour. Lorsque le comptage arrive au bout, le compteur déclenche le signal EVENT 2.

4. Lorsqu'il s'agit d'une règle composée comme :

si <expression> alors <action élémentaire> après N milisec.

CONDITION — CONDITION TEMPORELLE

Il sera construite une horloge logicielle (ou un compteur d'événements dans le cas d'une condition de type 3) auquel il sera associé un événement EVENT 3. Cette interruption logicielle sera déclenchée lorsque le comptage arrive au bout. La règle sera, dans le cas, transformée et mémorisée comme :

si <expression> et EVENT 3 <action élémentaire>

### 3.2.3.3 *Traitement des actions élémentaires*

Nous ne traiterons ici que les actions qu'il est permi d'appliquer sur les tâches. Pour ceci nous donnons un petit rappel des états possibles d'une tâche. [cf. Fig. 6]

Les tâches actives, sont celles qui sont en possession de toutes ses ressources, sauf, l'unité centrale.

Les tâches inactives ou dormantes, sont des tâches qui ne travaillent qu'un court instant, et qui n'ont rien à attendre, sauf qu'on ait besoin d'elles.

Les tâches en attente, sont celles qui attendent l'arrivée d'un événement, ou qui sont en attente de messages.

La tâche en cours d'exécution, est celle qui possède la ressource unité centrale. Elle est élue parmi les tâches actives, par ordre de priorité. Toutefois, une demande explicite d'exécution donne la ressource unité centrale a une tâche qui n'est pas nécessairement la plus prioritaire.

Parmi les actions élémentaires que l'on peut réaliser sur les tâches, nous avons :

exécuter TASK 1 : demande l'exécution immédiate de la tâche TASK 1. La tâche en cours d'exécution est suspendue, même si elle est plus prioritaire, au profit de TASK 1.

activer TASK 2 : provoque le passage de la tâche TASK 2 à la pile des tâches actives.

desactiver TASK 2 : la tâche TASK 2 est chainée à la pile des tâches inactives ou dormantes.

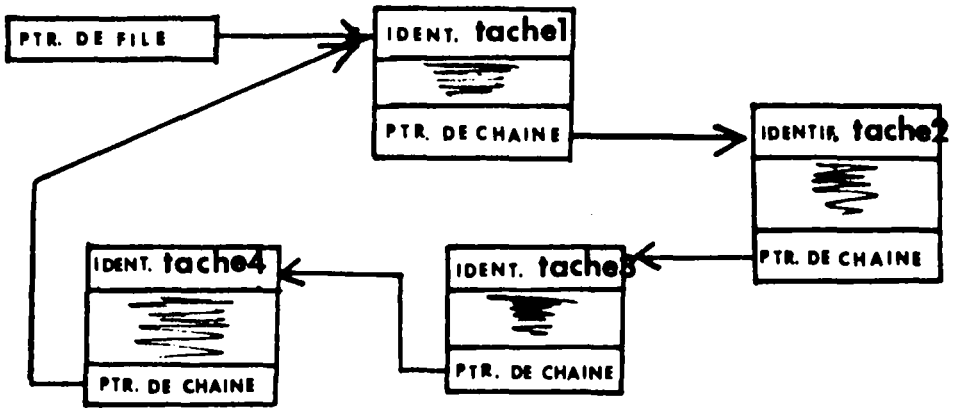
attendre TASK 2 : range la tâche TASK 2 dans la file des tâches en attente. Certaines tâches qui ont besoin pour s'exécuter d'avoir reçu toutes ses variables de connexion, seront chainées

dans la file des tâches en attente. Ceci à la fin de leur exécution.

D'autres primitives comme créer et détruire, ne sont utiles que lor d'une phase de mise au point que nous n'étudions pas pour le moment.

Lorsqu'il s'agit d'une action élémentaire isolée, où l'évaluation d règle de synchronisation n'entraine que le déclenchement d'une action élém taire, le module ECHANGE agit de la façon suivante :

si l'action élémentaire est : activer, desactiver, attendre, alors module ECHANGE positionne le bit correspondant, dans le mot d'état la tâche. Ensuite, il y a accès à la file respective, pour chainer tâche à une de ces files. Pour ceci chaque file a besoin d'un point vers l'une des tâches de la file.




PTR de file = Pointeur vers une des tâches de la file.
PTR de chaine = Pointeur vers la tâche suivante.

*FIG. 5 : CHAINAGE DES TACHES*

Par exemple, pour une inclusion de tâche, le pointeur est mis à jour comme suit (la tâche 5 est la tâche à inclure dans la file) :

Acc ← PTR-DE-FILE
PRT-DE-FILE ← identification TACHE 4
PRT-DE-CHAINE ← Acc

PTR DE CHAINE est un champ réservé au chainage des tâches dans les files. Ce champ est réservé dans le descripteur de chaque tâche. Il contient l'identification de la tâche (ident. TACHE).

Si l'action élémentaire est exécuter T2, par exemple, elle ne demande qu'à suspendre l'exécution de la tâche en cours. Il faut remarquer que le contrôle de l'unité centrale sera donnée à T2, si toutefois T2 appartient à la file des tâches actives.

Par contre, le problème des actions séquentielles et semi-parallèles, est laissé aux soins du concepteur, pour y apporter une solution. Nous rappellons que cette dernière situation survient, lorsque deux ou plusieurs actions peuvent être déclenchées simultanément. Ceci, lors de l'évaluation d'une ou plusieurs règles de synchronisation.

### 3.2.4 Descripteur de tâche.

Nous allons donner, ce qui pourrait être le descripteur de tâche. [INT 78, MIC 77]. Ce descripteur est construit lors de la compilation de chaque tâche. [cf. Fig. 7]

Descripteur de tâche K.

1. identification de la tâche

2. priorité de la tâche

3. Etat de la tâche

   - en exécution
   - active
   - en attente
   - drapeau de comptage de nombre d'exécutions
   - messages reçus

4. Vecteur d'échange : chaque fois qu'une variable de connexion est modifiée, le module ECHANGE positionne le bit correspondant à cette variable.

5. Zone des données qui constitue des données propres à la tâche

6. Zone programme : elle contient le début de l'ensemble d'instructions de la partie traitement.

7. Champ pile, c'est la zone de sauvegarde d'informations de la tâche.

8. LIEN d'appartenance aux files : il sert à chainer les tâches appartenant à une même file.

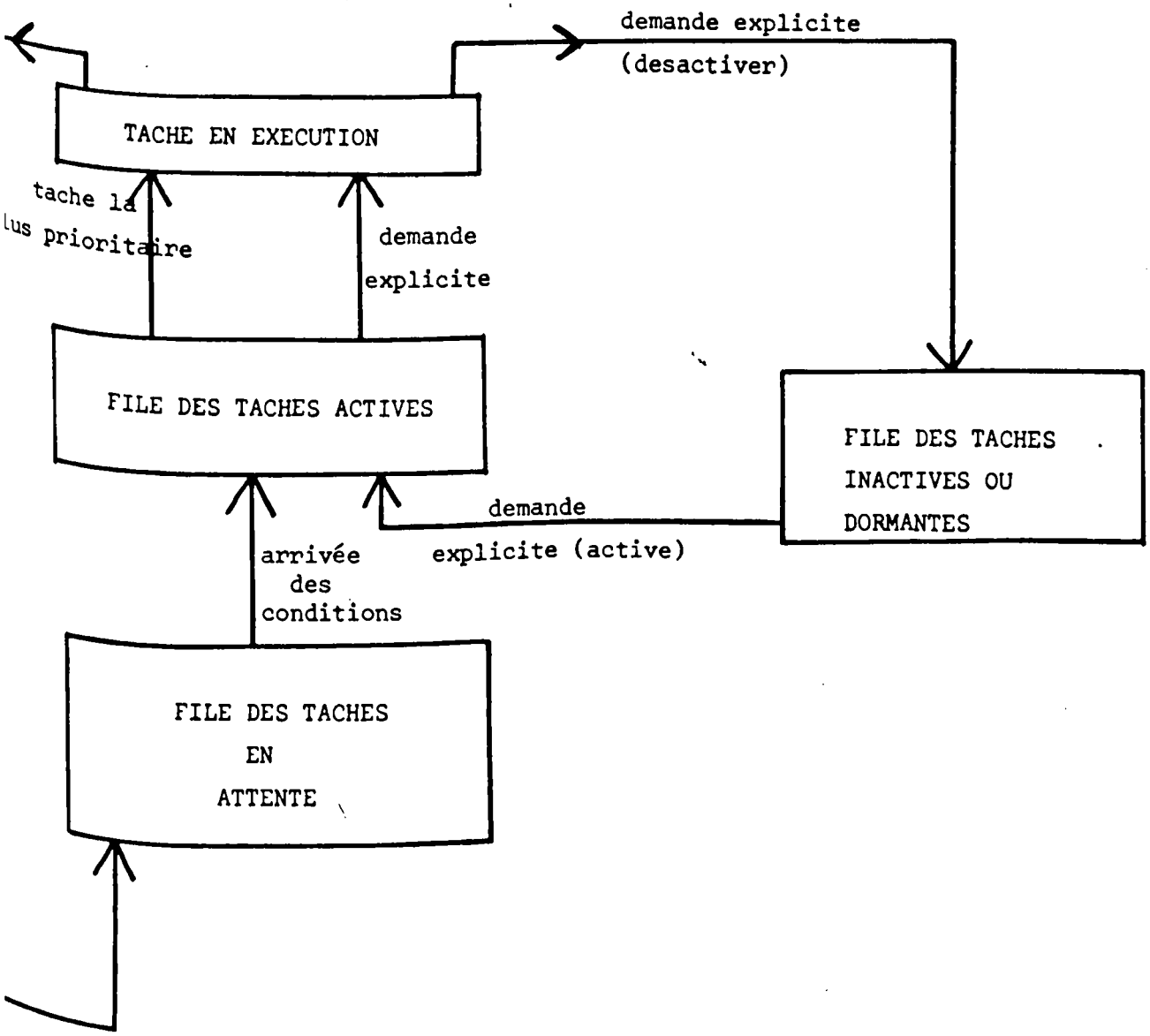
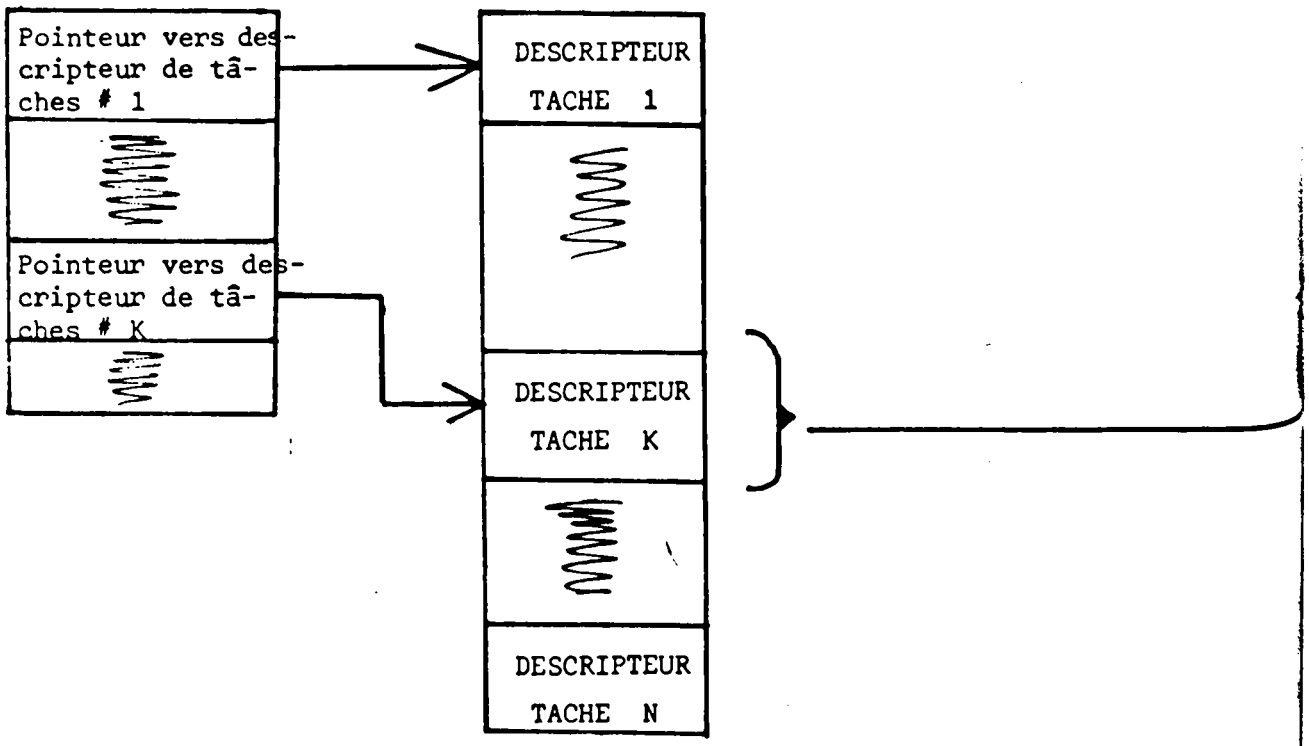



*FIG. 6 : SCHEMA D'EVOLUTION DES TACHES*

Nous pouvons donc définir le descripteur de tâche de la façon sui



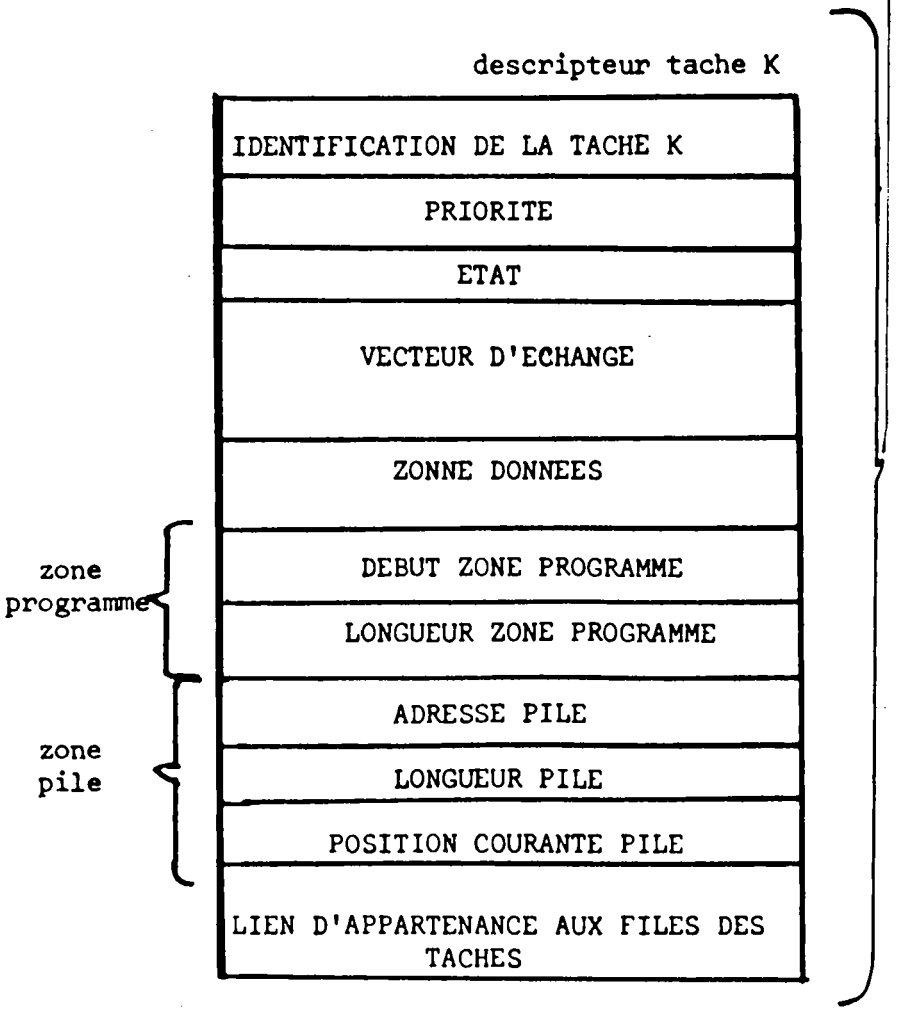






*FIG. 7 : DESCRIPTEUR DE TACHES*

A l'exécution, les structures d'échange et synchronisation sont exploitées de la façon suivante : lorsque une tâche produit un message, le module ECHANGE consulte la table des variables de communication. Toutes les tâches qui utilisent le message produit en seront averties.

Ainsi, le vecteur échange de chaque tâche réceptrice est mis à jour, en positionnant le bit correspondant au message produit. Si la tâche réceptrice à reçu tous les messages, dont elle avait besoin, alors elle sera mise à l'état actif.

Si le message produit est une variable de synchronisation, alors le vecteur de synchronisation est mis à jour. Selon la primitive d'échange utilisée, le contrôle du CPU sera donnée, à la tâche en cours ou à la tâche la plus prioritaire parmi les actives [Cf. 2. 3].

Nous donnons ci-après, la programmation des primitives envoyer et expédier. Ceci pour illustrer l'exploitation, à l'exécution, des vecteurs échange et synchronisation.

```
proc ENVOYER (VARIAB)
     début

(* VARIAB est cherchée dans la TABLE DE COMMUNICATION qui est composé de TABLE
 D'EVALUATION ET TABLE D'ECHANGE *)

          si VARIAB dans TABLE D'ECHANGE alors
          mémoriser <identification des tâches concernées>
          faire depuis 1 jusqu'à NB. TACHES CONCERNEES
          lire VECTEUR D'ECHANGE de <identification de tâche>
          RANG (VARIAB) de <identification de tâche> = 1

(* si la tâche n'attendait que ce message elle est activée *)

          mot d'état de <identification de tâche> = 1
          fin.faire
(* ensuite la tâche la plus prioritaire est invité à s'exécuter *)
```

choisir (tâche prioritaire)

(* VARIAB apparait dans la partie condition d'une règle de synchronisation *)

si VARIAB dans TABLE D'EVALUATION alors
mémoriser<numéro de règle de synchronisation>
évaluer règle de synchronisation

fin ENVOYER

La différence entre ENVOYER et EXPEDIER, réside dans le fait que EXPEDIER ne fait pas le choix parmi les tâches actives. Par conséquence, la tâche qui produit le message garde le contrôle de l'unité centrale.

## 3.5 MODULE INTERRUPTION

Manipuler les interruptions suppose, traiter les requêtes des usag et celles du matériel, concevoir les échanges, gérer la configuration, etc. En somme, il s'agit d'un accès fin aux possibilités du matériel. Ceci est un opération délicate, surtout, si comme nous le souhaitons, les concepts génér que nous developpons doivent rester indépendants de toute cible (ou matériel microprocesseur).

Parmi les interruptions que le noyau logiciel doit prendre en compt nous avons :

1. Interruptions internes

Ce sont des interruptions produites par des phénomènes inter au SBμP. Nous les appelons interruptions logicielles ou prog mées. Dans ce groupe nous classons les signaux d'interruptio émises par les tâches lors d'une demande de service.

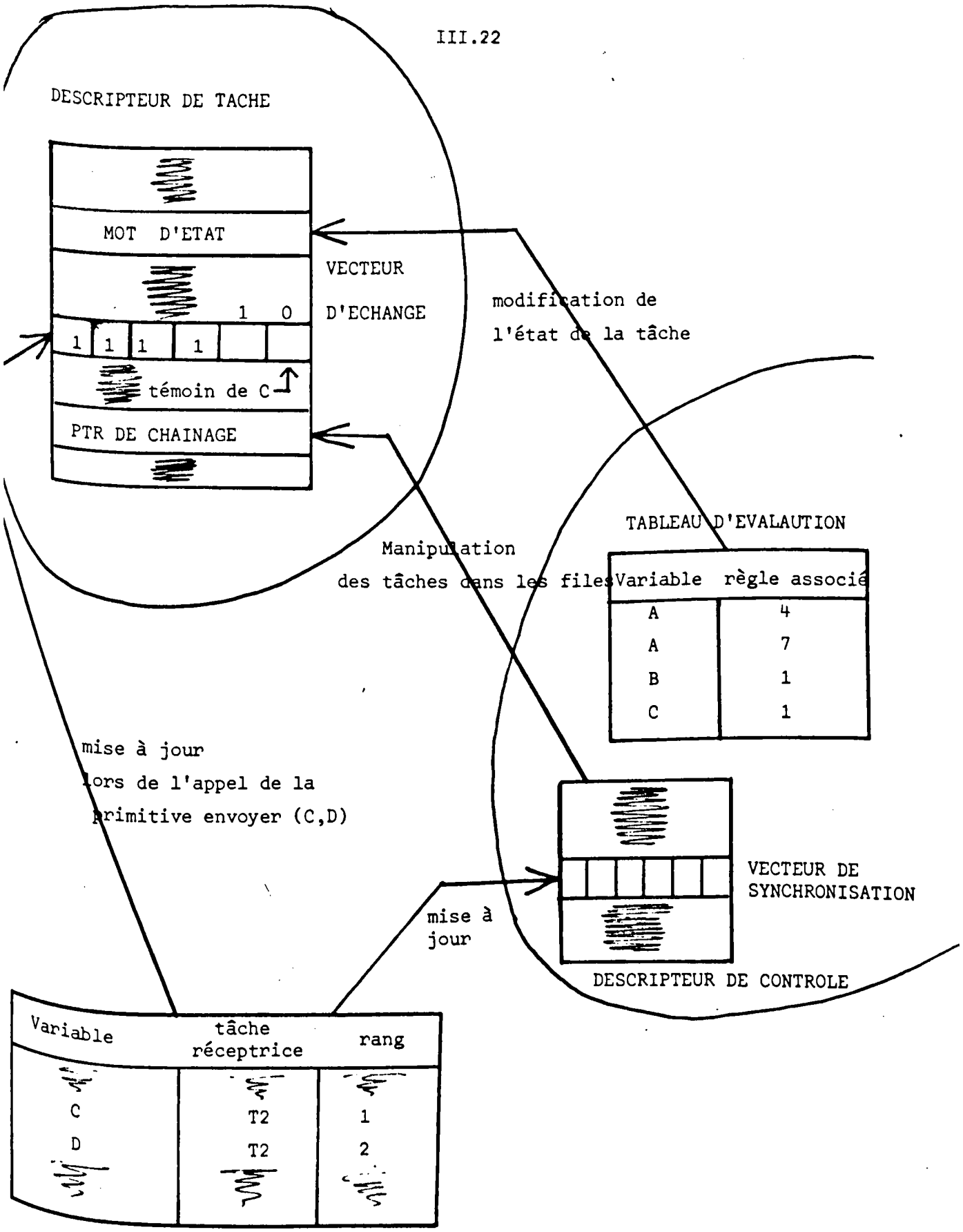



FIG. 8 RELATION ENTRE LES PARTIES ECHANGE ET SYNCHRONISATION

2. Interruptions en provenance de l'extérieur. Nous les appelons interruptions matérielles. Nous pouvons citer :

- demande de service en provenance des organes périphériques
- interruption horloge
- interruption due à un défaut d'alimentation
- redémarrage du système.

En général, une interruption demande un traitement très court. Lorsque ceci n'est pas possible, cette interruption demande l'exécution d'un traitement. Le traitement s'effectue cette fois sur une priorité plus basse.

Les interruptions logicielles ou programmées sont émises implicitement par les tâches, lors de l'appel des primitives d'échange. En effet, une primitive d'échange émet un signal d'interruption, qui provoque un branchement vers le module ECHANGE.

Qu'il s'agisse d'interruptions logicielles ou matérielles, la réponse à une interruption est presque toujours identique. A ceci presque, les interruptions matérielles ont toujours une priorité plus élevée que les interruptions logicielles. Nous pouvons résumer cette réponse comme suit [HIL 76] :

arrivée d' l'interruption
- le processeur envoi un accusé de réception
- sauver le contexte de la tâche en cours
- interdire, éventuellement, la prise en compte d'autres interruptions
- traitement de l'interruption
- restaurer le contexte de la tâche précédente

fin de traitement de l'interruption

En général, les microprocesseurs imposent des adresses d'implantation de certaines des routines de traitement des interruptions. Ainsi nous présentons ci-dessous quelques exemples.

| | | |
|---|---|---|
| µp 8085 | RESET | se branche à l'adresse 0000 |
| | TRAP | se branche à l'adresse 0024 |
| | RST 5.5 | se branche à l'adresse 0026 |
| | RST 6.5 | se branche à l'adresse 0034 |
| | RST 7.5 | se branche à l'adresse 0036 |

l'instruction RST branche, en plus, à 8 adresses

| | | |
|---|---|---|
| µp 6800 | $\overline{\text{RESET}}$ | se branche à l'adresse FFFE, FFFF |
| | $\overline{\text{NMI}}$ | se branche à l'adresse FFFC, FFFD |
| | instruction sw : | se branche à l'adresse FFFA, FFFB |
| | $\overline{\text{IRØ}}$ | se branche à l'adresse FFF8, FFF9 |

| | | |
|---|---|---|
| µp Z80 | $\overline{\text{NMI}}$ | se branche à l'adresse 0066 |
| | $\overline{\text{RESET}}$ | se branche à l'adresse 0000 |

L'utilisation de circuits decodeurs de priorité augmente sensiblement les adresses possibles de branchement des routines d'interruption.

La prétention d'un système utilisateur, est de pouvoir contrôler les opérations et les demandes de service adressés au µp. Ces opérations et demande de service, vont nécessairement varier d'un système à un autre. Pour cette raison, il serait souhaitable, que la portion gestion des interruptions, en général, soit implémentée comme des sous-routines, qui puissent être appellées par les tâches et modules. Ceci, de façon à pouvoir appeller des routines de gestion de ces interruptions, écrites par le concepteur. Cet approche permet, de personaliser un système utilisateur en tenant compte à chaque fois des périphériques présents dans l'application, sans disturber la logique des autres composants du système [SCE 78].

Pour ceci, l'adresse de début de ces routines est placée dans les vecteurs de celles qui apparaissent dans le corps du programme d'applicati Il ne reste au concepteur, que de substituer les appels des routines, par des routines de gestion particulières.

Pour chaque interruption en provenance des périphériques, COSMIC as cie une description comportant les éléments suivants [HEI 79] :

1. Un ensemble de positions mémoire propres à chaque périphériq Certains bits de ces registres servent à signaler des erreur et à contrôler le périphérique. En plus, ils peuvent conteni la taille des tampons et des informations pour les organes d DMA.

2. Un vecteur d'interruption par organe périphérique comportant l'adresse et la priorité du gestionnaire d'interruption du périphérique.

En résumé, les interruptions logicielles sont traités par le biais des primitives d'échange. Les interruptions matérielles exigent, par contr la description d'un certain traitement de la part du concepteur (routine d'interruption, gestionnaire périphérique, etc).
A partir des informations concernant les possibilités d'interruption du SBμP choisi, COSMIC integre, à la description du concepteur, les détails propres au système d'interruption du SBμP choisi.

## CONCLUSION

Nous avons présenté tout le long de ce chapitre, les actions du module ECHANGE lors de la transmission d'informations entre les tâches et lors du contrôle de l'évolution des tâches. Nous avons ensuite fait quelque remarques concernant la prise en compte des interruption. Ceci en ce qui concerne les spécifications fonctionnelles. Dans le chapitre suivant nous aborderons les spécifications technologiques.

# BIBLIOGRAPHIE


[HAN 79] HANNE J.E.
*Micros et moniteurs temps réel organisation et réalisation,*
MINIS et MICROS # 66

[HEI 79] HEIMBIGNER D.
*Writting device drivers in concurrent PASCAL,*
Operating system revue ACM, VOL 12, #4, oct. 1978

[HIL 76] HILBURN J.L, JULICH P.M
*Microcomputers/Microprocessors : Hardware, software and applications,*
Prentice Hall, 1977

[INT 78] INTEL
*RMX/80 users Guide*
Brochure #9800522 B, 1978

[LIS 78] LISTER A.M
*Principes fondamentaux des systèmes d'exploitation,*
Ed. Eyrolles, 1977

[MIC 77] MICROWARE SYSTEMS CORPORATION
*RT/68 MX system manual*
Brochure #RT68MXM, 1978

[SCE 78] SCELBI
*Scelbi's 8080 standard monitor,*
SCELBI computer consulting, 1978

# CHAPITRE IV : SPECIFICATIONS TECHNOLOGIQUES

INTRODUCTION

4.1 DISPOSITIFS D'ENTREE / SORTIE

4.2 DESCRIPTION DES PERIPHERIQUES

4.3 MODULE GENERATEUR D'INTERFACES

4.4 ACTIONS DU MODULE GENERATEUR D'INTERFACES

CONCLUSION

# INTRODUCTION

Un micro-ordinateur a besoin d'organes périphériques d'E/S pour communiquer avec l'extérieur. Nous faisons l'hypothèse, que le concepteur connait les caractéristiques fonctionnelles des organes périphériques nécessaires à la résolution du problème de l'utilisateur. Dans ce cas, COSMIC aide le concepteur à définir les liens entre le(s) périphérique(s), dont-il a besoin, et les circuits d'interface du SBμP.

La démarche que nous allons utiliser dans ce chapitre, est la suivante : nous étudions des caractéristiques générales de quelques dispositifs d'interface. Nous donnons après, un exemple de comment définir une configuration fonctionnelle en fonction d'un périphérique. A partir de là, nous dégageons un ensemble de caractéristiques à retenir, concernant les périphériques. Nous étudions ensuite,la façon d'utiliser les informations apportées par le concepteur. Pour finir, nous proposons une description standard, qui facilite la prise en compte des données sur les périphériques.

# INTRODUCTION

Un micro-ordinateur a besoin d'organes périphériques d'E/S pour communiquer avec l'extérieur. Nous faisons l'hypothèse, que le concepteur connait les caractéristiques fonctionnelles des organes périphériques nécessaires à la résolution du problème de l'utilisateur. Dans ce cas, COSMIC aide le concepteur à définir les liens entre le(s) périphérique(s), dont-il a besoin, et les circuits d'interface du SBμP.

La démarche que nous allons utiliser dans ce chapitre, est la suivante : nous étudions des caractéristiques générales de quelques dispositifs d'interface. Nous donnons après, un exemple de comment définir une configuration fonctionnelle en fonction d'un périphérique. A partir de là, nous dégageons un ensemble de caractéristiques à retenir, concernant les périphériques. Nous étudions ensuite,la façon d'utiliser les informations apportées par le concepteur. Pour finir, nous proposons une description standard, qui facilite la prise en compte des données sur les périphériques.

## 4.1 DISPOSITIFS D'ENTREE / SORTIE

Un dispositif d'E/S[1], est un circuit dont la fonction principale, est celle de servir de véhicule de transit des informations entre le microprocesseur et la périphérie (Fig. 1). Lorsque la configuration fonctionnelle du dispositif d'E/S, peut être modifié par des instructions issues du microprocesseur, nous parlons de dispositif programmable d'E/S.

Remarque : Nous rappelons que Système à Base de Microprocesseur (SBμP), est l'ensemble {microproc., mémoires, circuits annexes}. Ceci englobe, aussi bien les cartes offertes par le constructeur (SBC d'INTEL, MICRO-MODULES de MOTOROLA, etc.), que le système à base des circuits LSI élémentaires CPU, RAM, ROM, I/O, etc. L'un ou l'autre de ces deux ensembles est choisi par le concepteur à l'aide de COSMIC.

Schématiquement, un dispositif d'E/S peut être vu comme un ensemble de registres accessibles par le microprocesseur (Fig. 1). Tout périphérique peut donc être relié à un SBμP, moyennant une programmation adéquate des registres du dispositif d'E/S [INT 76, AMI 75]. Les fils reliés au dispositif d'E/S peuvent être de deux types :

1. Ceux qui sont utilisés pour communiquer avec le SBμP. Ceci par l'intermédiaire des bus des données, de contrôle et d'adresse.

2. Ceux qui sont utilisés pour communiquer avec la périphérie. Il s'agit des fils des données et des fils de contrôle.

Vis-à-vis de la description de l'application, ce sont les lignes qui permettent la communication avec la périphérie qui présentent l'intérêt principal. Pour cela, nous supposons que pour chaque dispositif d'E/S, il existe une description dans la base de données[2].

---

(1) ou dispositif d'interface ou coupleur d'E/S.

(2) au même titre que pour les autres circuits LSI de chaque microprocesseur ou pour chaque carte.

Il appartient donc à COSMIC, de trouver la configuration du dispositif d'E/S, adapté au périphérique à relier au SBuP.

Nous donnons ci-après, le diagramme synoptique d'un dispositif programmable d'E/S [INT 75] le 8255 d'INTEL (PPI) (Fig. 2).

Nous l'utiliserons par la suite, pour présenter la démarche de COSMIC une fois la description de l'application faite.

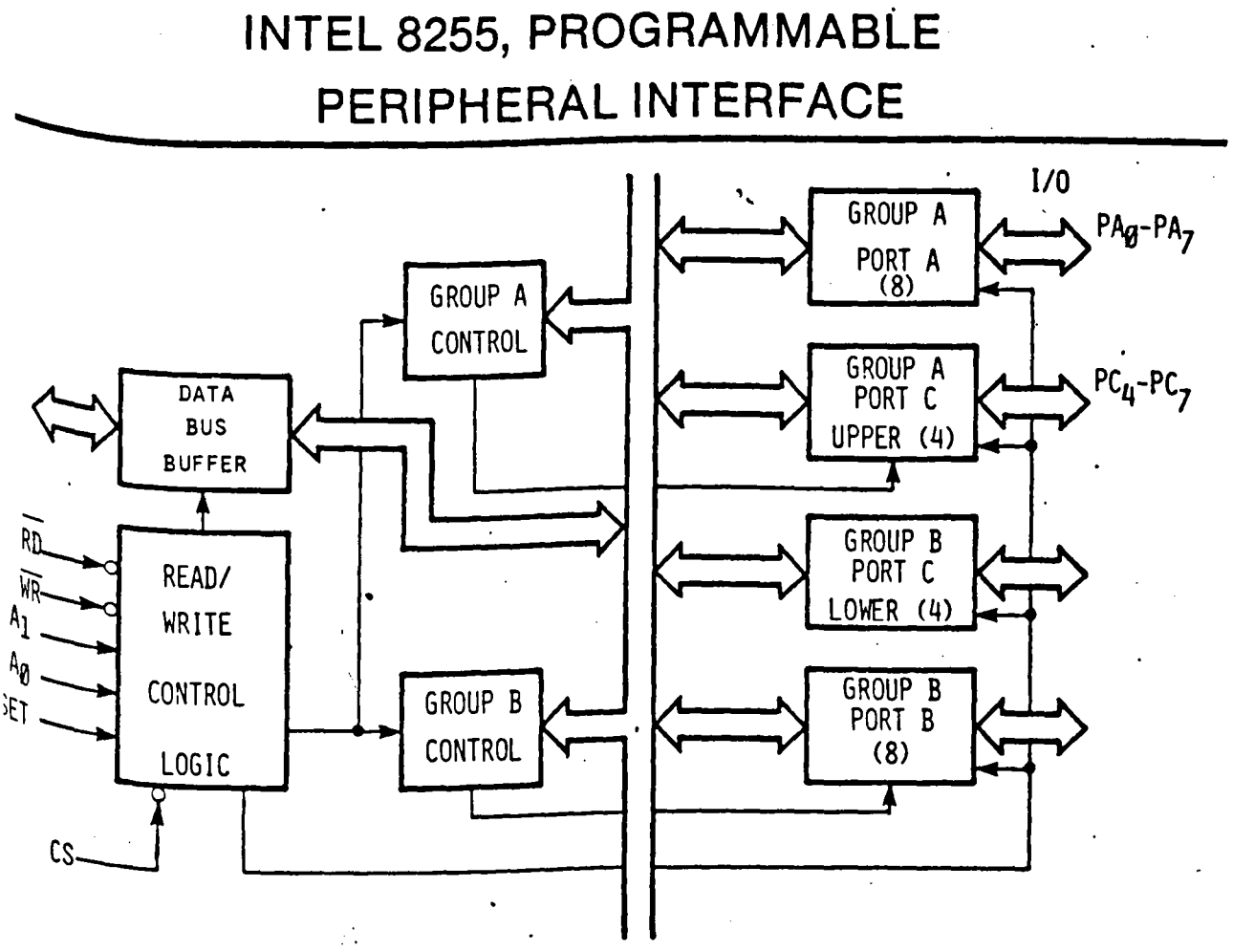



Fig. 2 : Brochage et organisation du coupleur parallèle (PPI) : 8255.

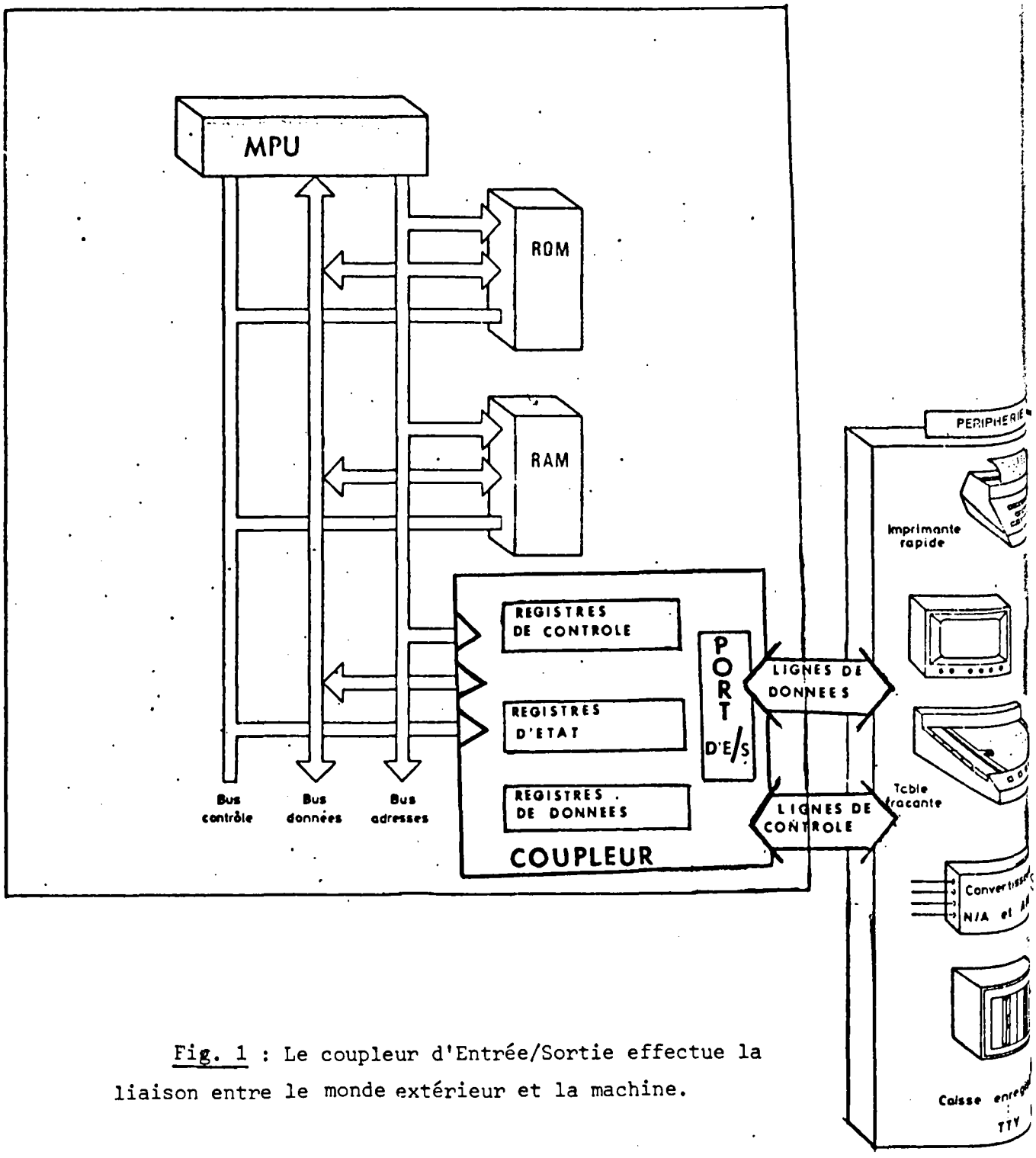



Fig. 1 : Le coupleur d'Entrée/Sortie effectue la liaison entre le monde extérieur et la machine.

Il appartient donc à COSMIC, de trouver la configuration du dispositif d'E/S, adapté au périphérique à relier au SBµP.

Nous donnons ci-après, le diagramme synoptique d'un dispositif programmable d'E/S [INT 75] le 8255 d'INTEL (PPI) (Fig. 2).

Nous l'utiliserons par la suite, pour présenter la démarche de COSMIC une fois la description de l'application faite.

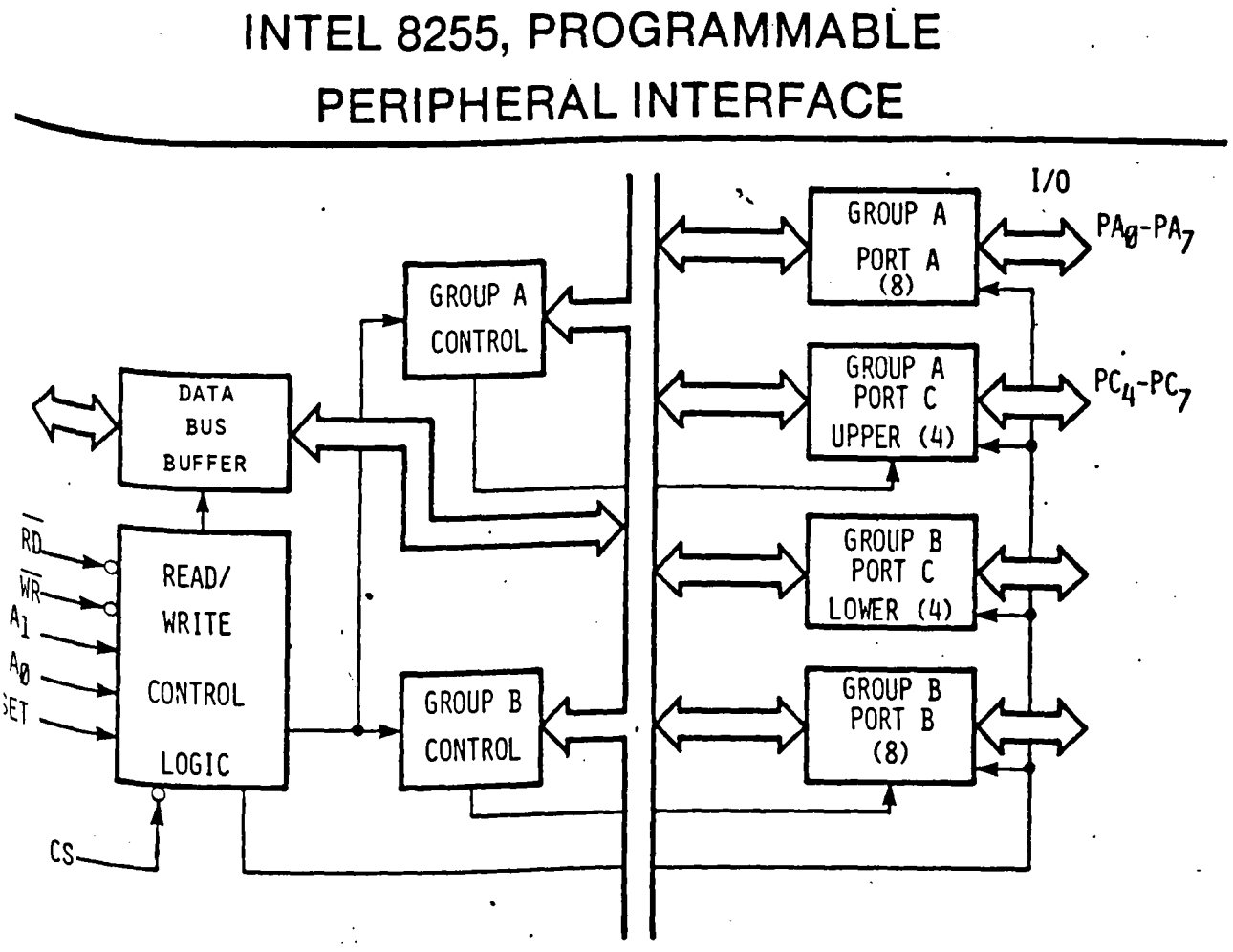



Fig. 2 : Brochage et organisation du coupleur parallèle (PPI) : 8255.

## 4.2 DESCRIPTION DES PERIPHERIQUES

COSMIC doit posséder une description standard des dispositifs d'interface, et la façon d'initialiser ces registres, en vue d'assigner un rôle aux ports d'E/S. Ceci, de façon à pouvoir chercher automatiquement la configuration fonctionnelle la mieux adaptée à chaque périphérique.

Un périphérique est un appareil qui sert à fournir des informations depuis l'extérieur au SBµP. Ou bien permet au SBµP à communiquer des informations à l'extérieur. Parmi les fils reliés au périphérique nous distinguons :

1. Ceux qui sont propres à la communication avec le SBµP et que nous appelons lignes d'E/S.

2. Ceux qui sont utilisés pour des fonctions spéciales : alimentation, masse, signaux horloge, etc.

Nous ne nous intéressons qu'au premier groupe de fils.

L'assignation du rôle des ports d'E/S du circuit d'interface, nous gère une classification de ces lignes d'E/S qui seront reliées au SBµP en

1. LIGNES SHAKEHAND (ou lignes d'accusé de réception) qui assurent la synchronisation du dialogue des deux unités interconnectées. Par exemple : STROBE, ACKNOWLEDGE, aussi bien que RTS, CTS, DSR, DTR dans une interface MODEM.

2. LIGNES DE COMMANDE (command lines) dont le rôle est le déclenchement des fonctions périphériques ou la communication de l'état du périphérique.

3. LIGNES DE DONNEES (data lines) qui assurent la transmission des données entre le µP et le périphérique.

En classant les lignes d'E/S de chaque périphérique en HANDSHAKE LINES, COMMAND LINES et DATA LINES, le concepteur permettra à COSMIC une assignation plus aisée du rôle des ports d'E/S, du dispositif d'interface.

Voici un exemple ou nous avons caractérisé l'ensemble des lignes d'E/S du périphérique CLAVIER-AFFICHAGE. [ALT 76].

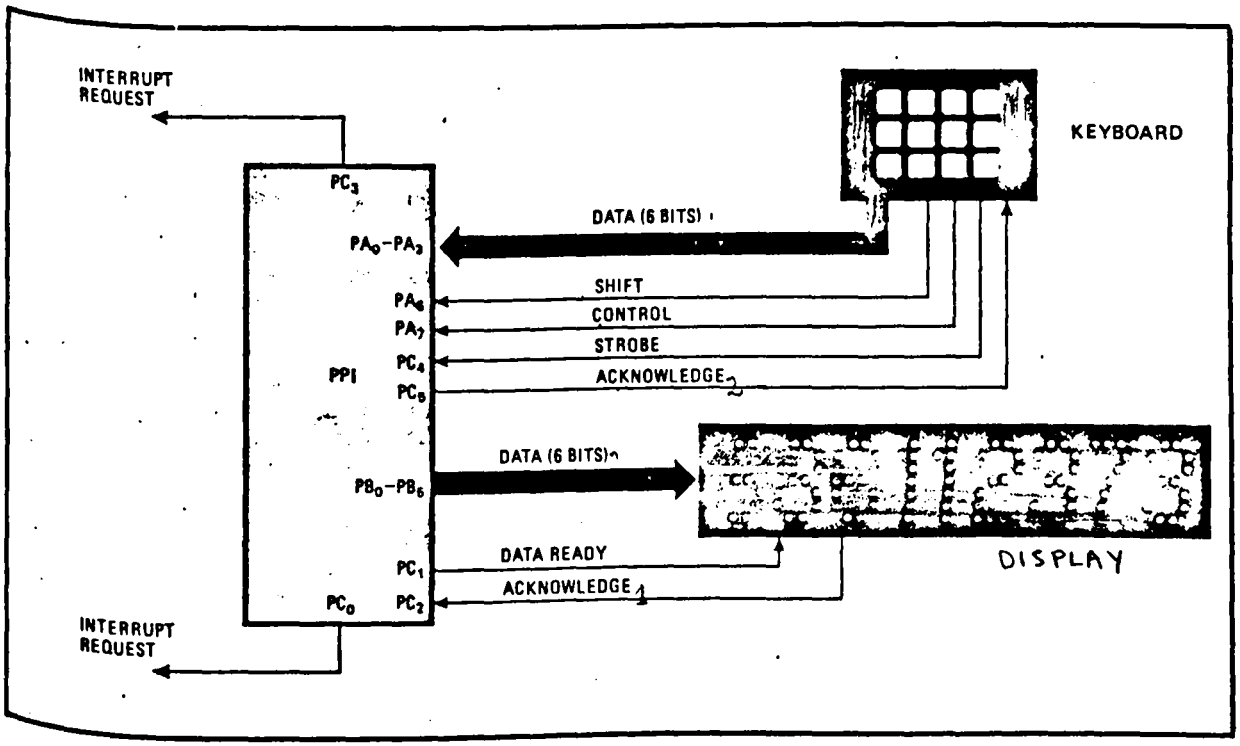



Fig. 3 : Keyboard and display. Exemple de caractérisation des lignes périphériques.

Dans la table ci-dessus nous donnons la caractérisation des différentes lignes de communication entre périphérique et dispositif d'interface de la figure 3 :

| SHAKEHAND | COMMAND | DATA |
|---|---|---|
| STROBE | SHIFT | $DATA(6)_1$ |
| $ACKNOWLEDGE_1$ | CONTROL | $DATA(6)_2$ |
| DATA READY | | |
| $ACKNOWLEDGE_2$ | | |

Table 1 : Caractérisation de lignes de la figure 3.

Ce qui intéresse COSMIC, est donc, l'information qui permettra :

1. De choisir les circuits d'interface périphériques : série/parallèle, choix parmi les série/parallèle, (s'il y en a plusieurs).
2. d'Assigner des rôles aux ports d'E/S du circuit d'interface.

L'utilisateur devra donc fournir deux types de renseignements :

1. La configuration fonctionnelle des lignes en E/S de chaque périphérique.

2. La relation logique entre les variables. Ce traitement sur les variables peut encore se diviser, dans un bon nombre de cas, en deux ensembles de traitement :

   - Le gestionnaire périphérique qui décrit le traitement à effectuer de façon à traiter une donnée élémentaire.

   - Le traitement proprement dit sur l'ensemble des données, ou la manipulation d'un bloc de données en Entrée ou en Sortie.

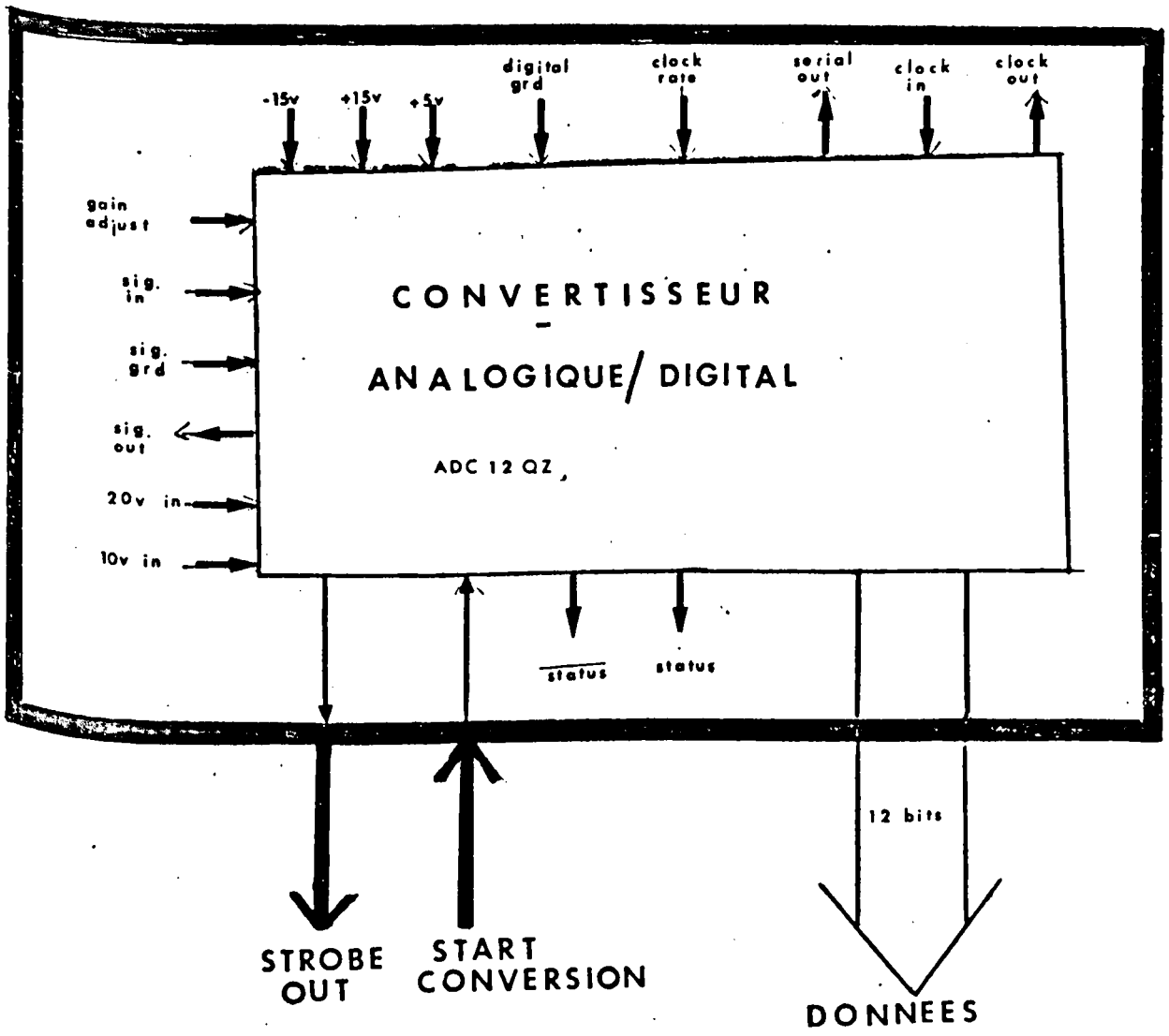



Fig. 4 : Schéma synoptique du périphérique convertisseur [ANA 78]

Nous allons montrer quelle est la démarche à suivre par le concepteur, lors de la description des périphériques. Pour cela nous allons décrire un des périphériques de l'exemple de la fabrique de lard dont le schéma synoptique apparaît à la figure 4. Il faut remarquer que tous les autres périphériques doivent aussi l'être ou, en tout cas, référencés, de façon à puiser leur description de la base des données.

Nous donnons ensuite, dans la partie spécifications technologiques le détail des lignes en E/S de chaque périphérique. Le gestionnaire périphérique et le traitement de l'ensemble des données, seront finalement décrits en tant que tâches dans la partie opérative.

Remarque : Pour le périphérique que nous décrivons, la transmission des bits des données se fait en parallèle. Lorsque la transmission des données est série, le concepteur devra décrire des caractéristiques supplémentaires propres à la transmission série.

```
périphérique CONVERTISSEUR

vitesse 40 MICROSECONDES

capacité

gestionnaire CONVERSION


début

    shakehand sortie STROBE.OUT booleen

    commande entrée START.CONVERSION booleen

    données sortie VALEUR : tableau [1..12]
                            booleen

fin CONVERTISSEUR
```

Le vecteur d'interruption [cf. 3.2.5], est substitué par la description de la tâche gestionnaire.

Ainsi, le gestionnaire périphérique (tâche CONVERSION) peut être décrite de la façon suivante:

```
tâche CONVERSION

    déclaration des variables
```

```
début

      START. CONVERSION <— 1

      écrire CONTROL.PORT

      START. CONVERSION <— 0

      écrire CONTROL.PORT

      attendre STROBE.OUT

      lire DATA.PORT

      VALEUR <——— DATA.PORT

      mémoriser VALEUR

      envoyer STROBE

fin CONVERSION
```

Lorsque nous parlons alors de CONTROL.PORT, DATA.PORT, etc. il s'agit de manipuler les lignes de COMMANDE, des DONNEES, etc. correspondant au périphérique. Ainsi,

SHAKEHAND.PORT corresponds aux lignes SHAKEHAND
CONTROL.PORT corresponds aux lignes de COMMANDE
DATA.PORT corresponds aux lignes de DONNEES

Si le périphérique est un standard, alors le gestionnaire périphérique est un standard. Dans ce cas, COSMIC prend la description fonctionnelle du périphérique d'un catalogue de périphériques standard. Ce catalogue apparaît dans la base des données consultée par COSMIC.

La description alors d'un tel périphérique se fera de la façon suivante :

périphérique CONVERTISSEUR

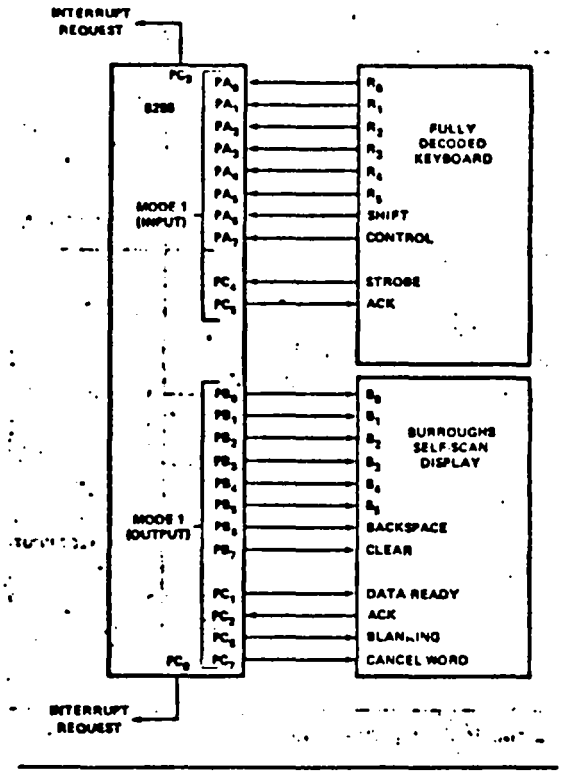
VITESSE :

CAPACITE :

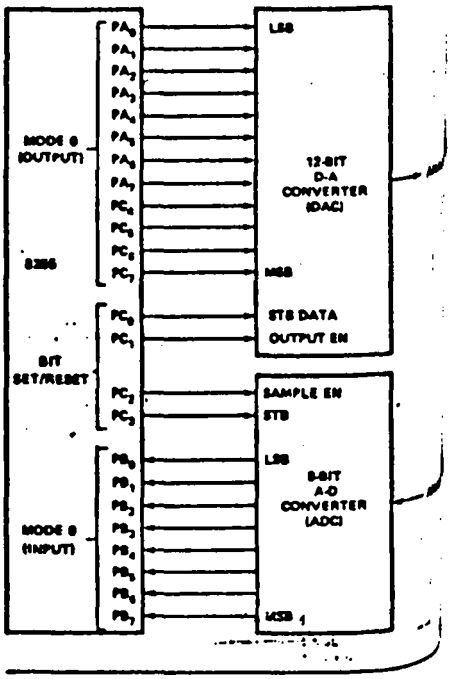
gestionnaire : CAD 1

fin CONVERTISSEUR

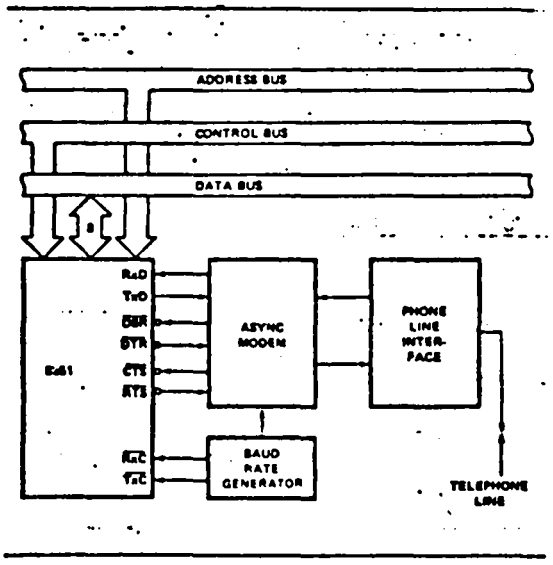
Il est certain que, dans ce cas, la description de variables dans la tâche CAD-1, doit correspondre à la définition des variables du périphérique CONVERTISSEUR, dans la base des données.



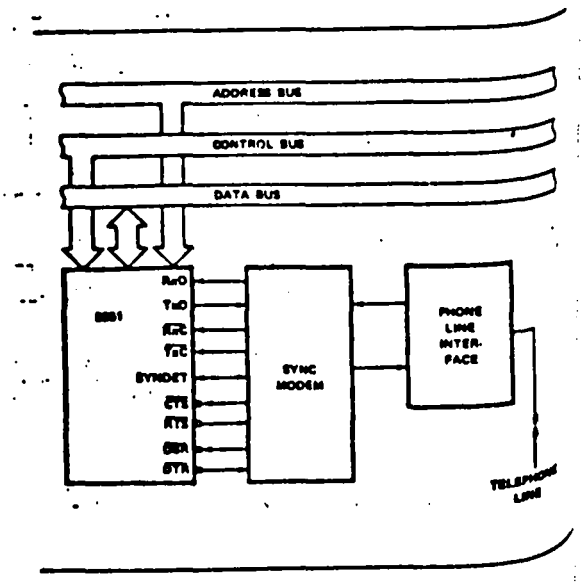

A. INTERFACE CLAVIER/AFFICHAGE




B. INTERFACE A/D ET D/A




C. INTERFACE ASYNCHRONE A UNE LIGNE TELEPHONIQUE




D. INTERFACE SYNCHRONE A UNE LIGNE TELEPHONIQUE



Fig. 5 : Configurations périphériques standard [INT 75]

## 4.3 MODULE GENERATEUR D'INTERFACES

Une fois décrites complètement l'ensemble de spécifications fonctionnelles, et technologiques, COSMIC construit des structures de données, nécessaires à la résolution du problème de l'utilisateur. Nous appelons MODULE GENERATEUR D'INTERFACES le module de COSMIC qui se chargera, le moment venu, de manipuler les données des spécifications technologiques. Ceci, en vue d'assigner le rôle aux ports d'E/S du dispositif d'interface, adapté à chaque périphérique de l'application.

En fonction des renseignements fournis par le concepteur, le module générateur d'interfaces, construit ce que nous appelons de MOT DES DONNEES. Le mot des données contient les caractéristiques de transmission propres à chaque périphérique :

1. Description des lignes d'E/S du périphérique.

2. Des renseignements concernant la vitesse de transmission et la capacité du stockage.

3. L'identification de la tâche qui se charge de la gestion du périphérique.

4. Des détails concernant le mode de transmission : parallèle/série, synchrone/asynchrone, etc.

Si la transmission est parallèle, le mot des données est constitué par :

a) Définition et description des lignes des données en E/S.

b) Définition et description des lignes de contrôle en E/S.

c) Vitesse de transmission et capacité de stockage.

d) Le nom de la tâche gestionnaire.

Si par contre la transmission est série, alors le mot des données contiendra les renseignements suivants :

1. Désignation des lignes de données en E/S.

2. Définition et description des lignes de contrôle d'E/S

3. Type de transmission : synchrone/asynchrone.

4. Mode de transmission : interne/externe.

5. Nombre de caractères de synchronisation : 1/2.

6. Caractères de synchronisation.

7. Rapport de transmission : ÷ 1 / ÷ 16 / ÷ 64.

8. Format des caractères : 5/6/7/8 bits.

9. Control de parite : oui/non.

10. Bits stop : oui/non.

11. Nombre de bits stop : 1, 1 1/2, 2.

12. Conditions spéciales d'activation de certains bits de contrôle.

Cette description sera faite à l'aide du langage de définition.

## 4.4 ACTIONS DU MODULE GENERATEUR D'INTERFACES

Les actions du module se situent après la description complète de l'application, et après la phase de choix du matériel microprocesseur. Nous allons distinguer ces différentes actions, en nous aidant de la description faite du périphérique CONVERTISSEUR dans la partie SPECIFICATIONS TECHNOLOGIQUES.

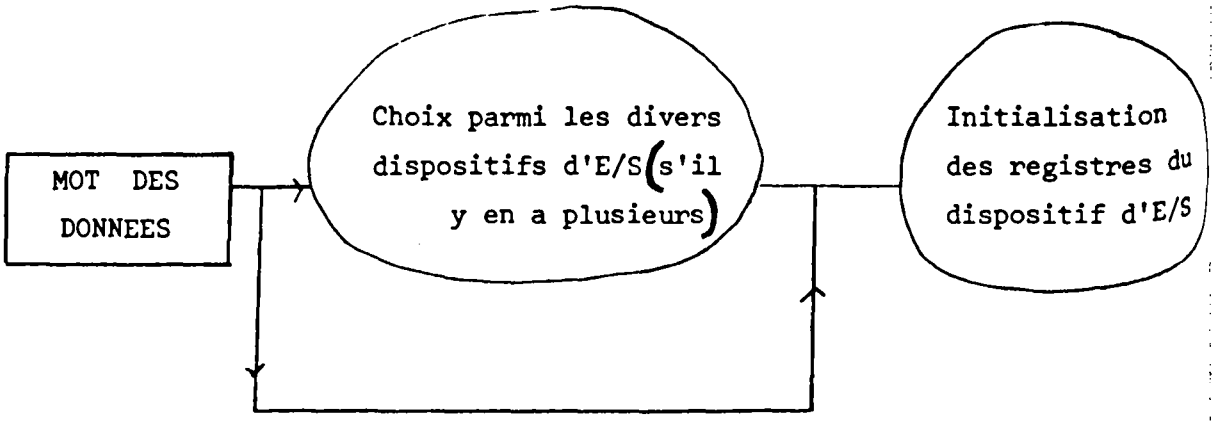
Une fois que le mot des données est défini, COSMIC va l'utiliser de deux façons :

1) Comme un critère de choix. Ceci pour éliminer des incompatibilités grossières entre une famille de circuits d'interface et la configuration fonctionnelle donnée du périphérique, lors de la phase de choix du matériel microprocesseur.

2) Comme une donnée de l'algorithme d'initialisation des registres du dispositif d'E/S, correspondant à chaque périphérique (ou groupe de périphériques).

La phase de choix du dispositif d'interface adapté au périphérique, est fortement influencée par le format de transmission : série ou parallèle. En général, il existe toujours un dispositif d'interface série ou parallèle appartenant à la famille microprocesseur choisie. S'il en existent plusieurs avec des caractéristiques voisines (par exemple le Z 80 - PIO et Z 80 A - PIO) le concepteur, aidé par COSMIC, choisira l'un ou l'autre d'entr-eux le mieux adapté à la fonction attendue. Le choix sera donc fait en fonction des renseignements du mot des données COSMIC.

Le schéma, pour chaque périphérique, sera le suivant :




Si l'option construction du microordinateur à partir de boîtiers est retenue, il faudra choisir parmi les circuits d'interface (s'il y en à plusieurs) celui qui s'adapte le mieux à la configuration du périphérique. Il est intéressant de laisser un niveau de choix au concepteur dans cette étape.

Par contre, si l'option carte assemblée (SBC d'INTEL, MICROMODULES MOTOMOLA, etc.) est retenue, il ne nous reste que l'initialisation des circuits d'interface. Ceci car, généralement, la carte assemblée possède ses circuits d'interface d'E/S.

Pour assigner le rôle aux ports d'E/S du dispositif, il faut connaître outre la configuration fonctionnelle du périphérique, l'algorithme d'initialisation de chaque dispositif d'interface.

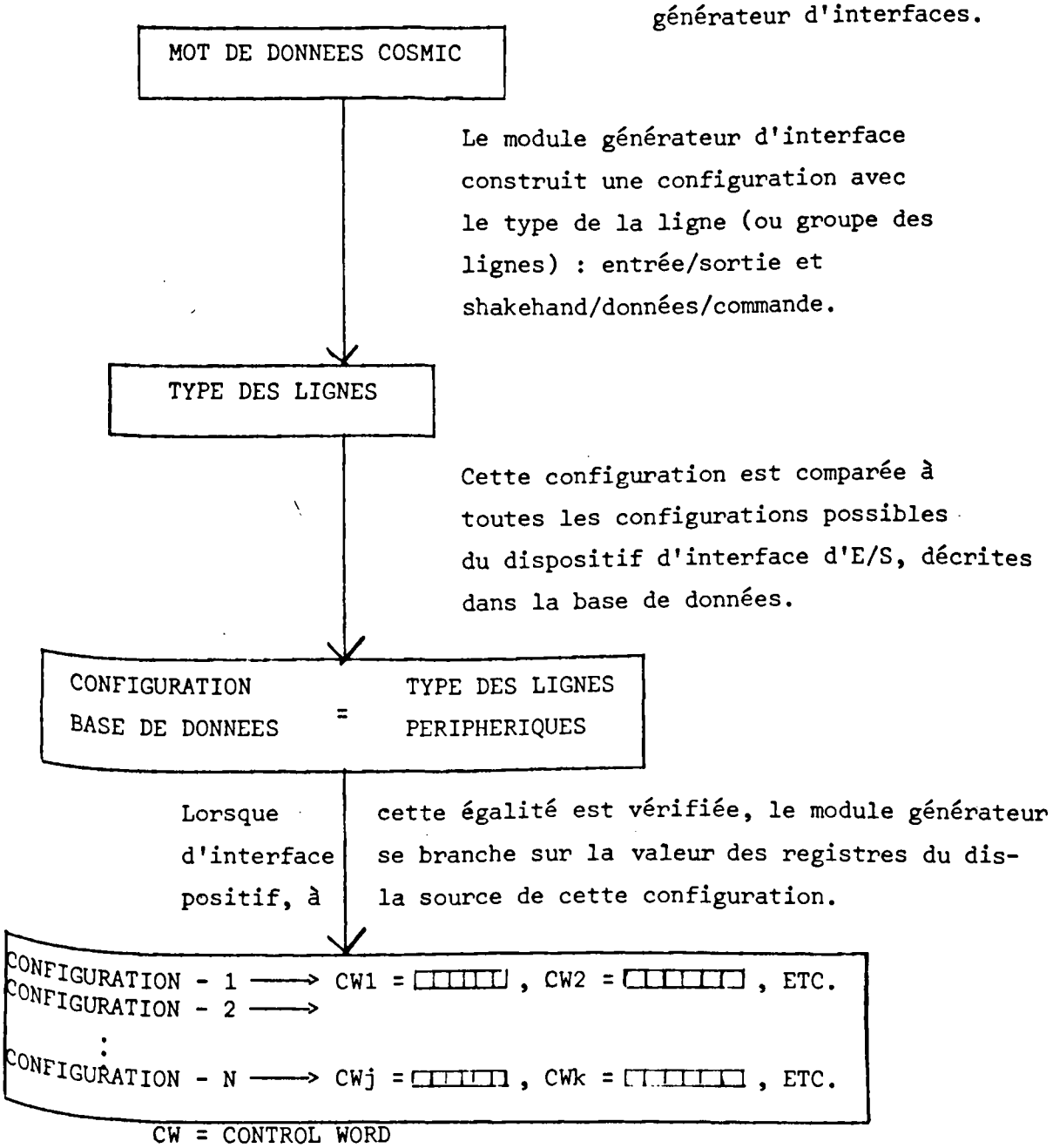
C'est ainsi, que nous décrirons dans la base des données technologique, toutes les configurations possibles de chaque dispositif d'E/S. A partir du mot de données, le module générateur d'interfaces, extrait les renseignements qui permettent d'initialiser les registres du dispositif d'interface. COSMIC construit, de cette manière, une configuration avec les informations nécessaires à l'assignation du rôle des ports d'E/S, c'est-à-dire

a) Désignation de la ligne (ou groupe des lignes) comme étant d'entrée ou de sortie.

b) Désignation de la ligne (ou groupe des lignes) comme appartenant à l'un des trois groupes suivants : lignes shakehand, lignes des données, lignes de commande.

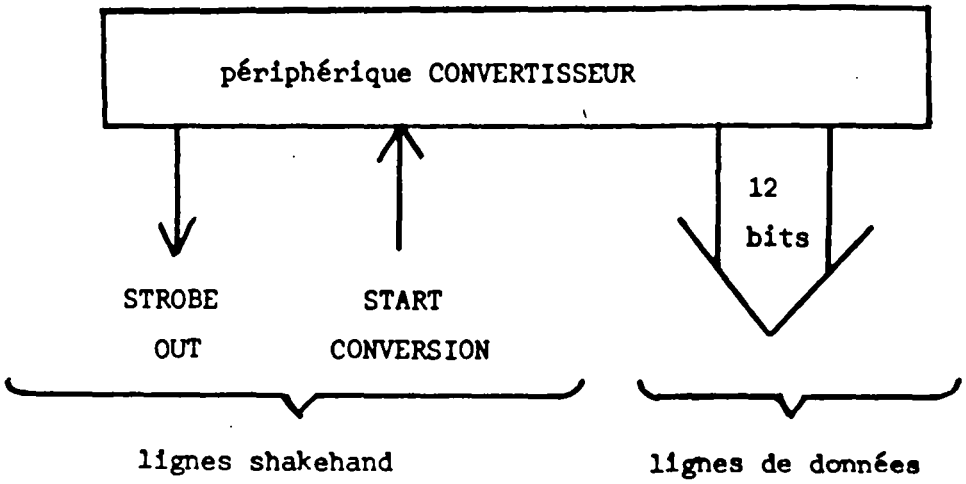
Le module générateur d'interfaces compare alors cette configuration avec celles décrites dans la base de données. Lorsqu'il en trouve une, il se branche alors à la suite d'opérations qui ont entraîné cette configuration.

Fig. 6 : Schéma de travail du module générateur d'interfaces.



Nous allons montrer, à partir de l'exemple du périphérique convertisseur, quelle est la démarche à suivre par le module générateur d'interfaces. Pour ceci, nous supposerons que le matériel microprocesseur choisi, dans une étape précédente, a été le 8080 d'INTEL. Ceci dit le choix du CPU entraîne le choix des circuits annexes associés au microprocesseur ou carte choisies.

La configuration fonctionnelle du périphérique est la suivante :




Comme le matériel microprocesseur que nous avons supposé choisi est le 8080 d'INTEL, alors le dispositif d'E/S que le générateur d'interfaces utilisera est le Parallel Peripheral Interface 8255 d'INTEL.

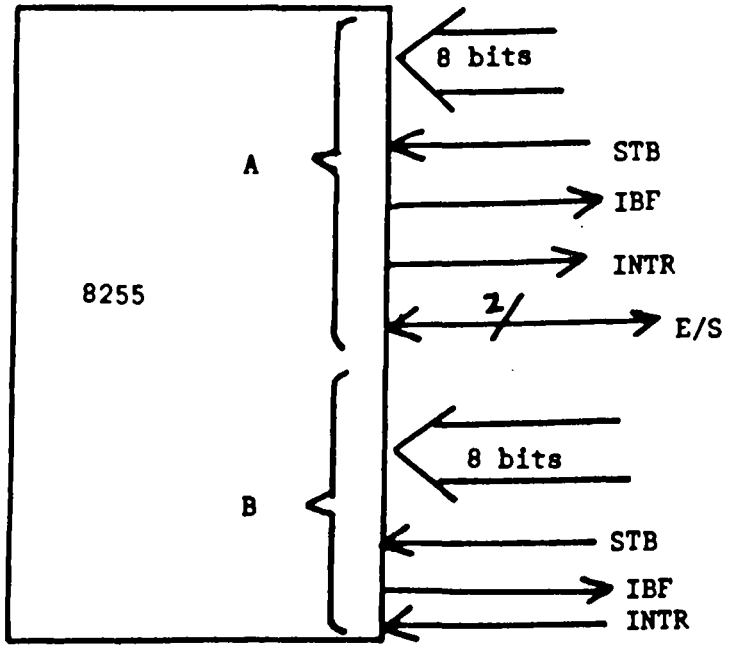
Remarque : S'il y a plusieurs dispositifs d'interface parallèle, le générateur d'interfaces doit utiliser celui de la famille, le mieux adapté au périphérique concerné.

On a donc, en général, les actions suivantes du module générateur d'interfaces :

1. Choix parmi les dispositifs d'E/S de celui le mieux adapté au périphérique. Dans notre exemple il n'y a qu'un seul dispositif : le 8255 d'INTEL.

2. Adressage éventuel des registres du dispositif d'E/S et initialisation des registres appropriés.

Parmi les configurations fonctionnelles du 8255, nous avons celle-ci :




Ce qui correspond vis-vis du $B\mu P$ à une configuration avec

16 bits de type données en entrée :
2 lignes shakehand en entrée : STB
2 lignes shakehand en sortie : IBF
2 lignes de commande : INTR
2 lignes programmables en I/O.

Cette configuration fonctionnelle correspond à une valeur des registres du dispositif programmable d'E/S égale à :

CW =

| 1 | 0 | 1 | 1 | X | 1 | 1 | X |
|---|---|---|---|---|---|---|---|

programmation port B — programmation port A

Ceci permettra d'utiliser 12, parmi les 16 lignes des données, pour recevoir les 12 bits des données en provenance du périphérique. La ligne IBF comme correspondante à la ligne du périphérique désignée START CONVERSION.

La ligne STB comme correspondante à la ligne STROBE.OUT. Le reste des lignes d'E/S du dispositif d'interface restent inutilisées.

L'exemple que nous avons montré avec des circuits programmables d'E/S parallèle, peut s'étendre à des circuits programmables d'E/S série. Ici la longueur du mot, le nombre de bits STOP, etc, sont des renseignements (à obtenir du concepteur) pour la programmation adéquate de ces circuits.

Dans chaque cas ce qui change est la programmation du circuit d'interface d'E/S spécifique à chaque famille µP. Nous avons pu programmer les différentes lignes d'E/S car nous connaissions le détail des lignes d'E/S du périphérique montré en exemple.

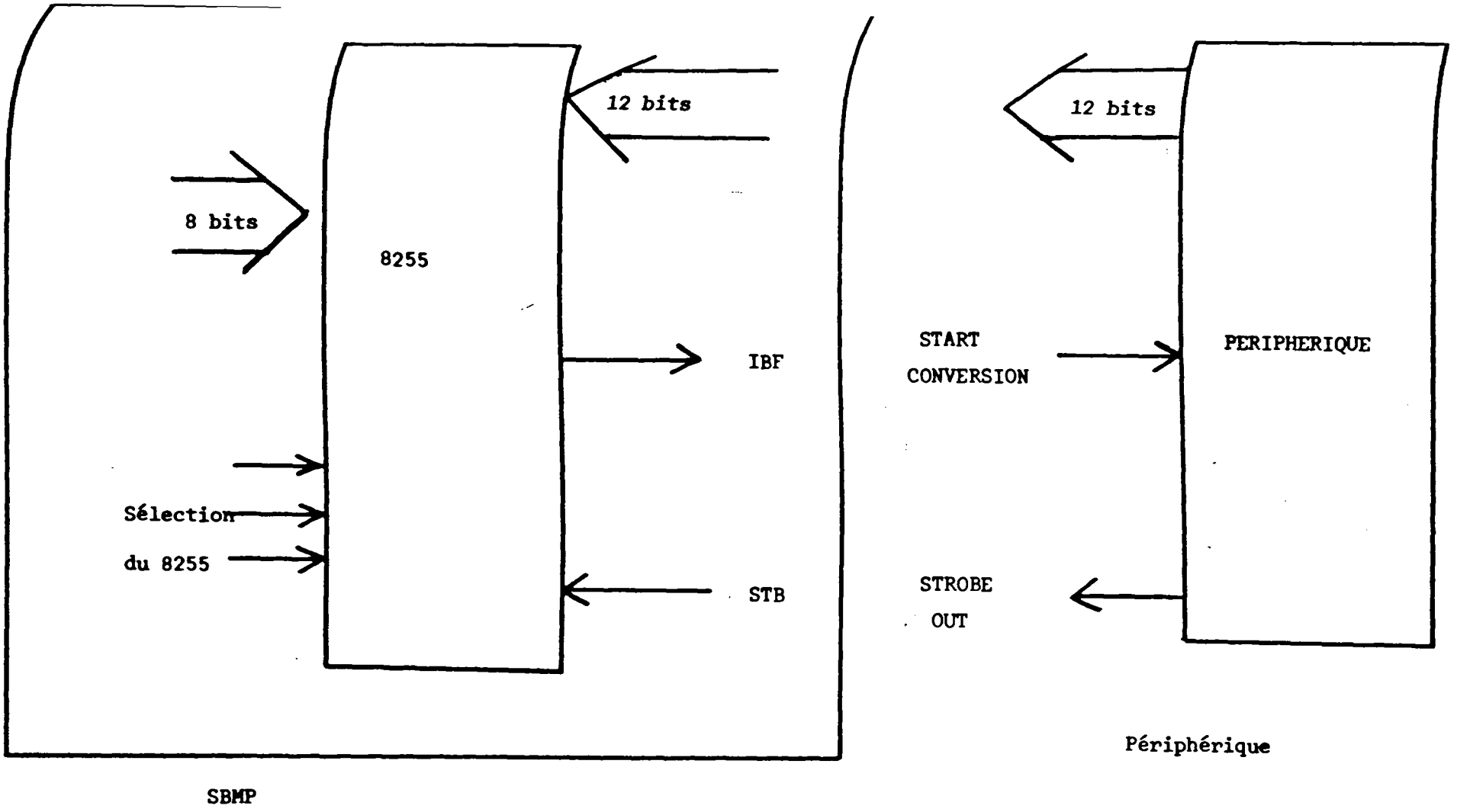





Fig. 7 : Correspondance des lignes d'E/S entre le périphérique et le dispositif d'E/S.

## CONCLUSION

Nous avons montré quelles sont les informations à décrire concernant les périphériques. Ceci donc culmine tout le long des chapitres 2, 3, 4 avec la description du problème de l'utilisateur. Dans le chapitre suivant nous étudierons plus particulièrement les problèmes d'estimation de temps d'exécution et taille mémoire requise pour les problèmes temps réel.

# BIBLIOGRAPHIE

[AMI 75] AMI
M 6800 data manual, 1975.

[ANA 78] ANALOG DEVICES
*Data acquisition product catalog*
Brochure ANALOG DEVICES.

[ALT 76] ALTMAN L. and SCRUPSKI S.
*Applying microprocessors*
ELECTRONICS MAGAZINE BOOKS SERIES, 1976.

[INT 76] INTEL
*Microcomputers peripherals user manual*
Brochure INTEL, 1976.

[INT 75] INTEL
*8080 microcomputer systems user manual*
Septembre 1975.

# CHAPITRE V :

# DÉCOMPOSITION D'UN $SB_\mu P$ EN MODULES FONCTIONNELS

INTRODUCTION

5.1 GENERALITES SUR LES MODULES MATERIELS DE BASE D'UN $SB_\mu P$

5.2 LES DIFFERENTS MODULES

- 5.2.1 Les bus
- 5.2.2 Les maitres
  - *5.2.2.1 L'ADM*
  - *5.2.2.2 Le MP*
- 5.2.3 Les esclaves
  - *5.2.3.1 Les mémoires*
  - *5.2.3.2 Les équipements d'E/S*

5.3 LES CONCEPTS DIRECTEURS POUR LA CONSTRUCTION DE LA BASE DE DONNEES

- 5.3.1 Ensemble de caractéristiques à retenir
- 5.3.2. Approche de description de la base

CONCLUSION

# INTRODUCTION

Jusqu'alors nous n'avons parlé que du problème de l'utilisateur, dont les éléments constituent le premier ensemble de connaissances, nécessaire au concepteur.

Nous allons parler du second ensemble de connaissances. Pour cela, nous choisissons de décomposer un $SB_{\mu}P$ en modules (matériels) fonctionnels. Cette façon d'aborder le problème nous permet de révéler à l'utilisateur les composants possibles de son futur système. Elle donne aussi la base pour l'utilisation de PYTHIE [PLE 79]

Pour décomposer un système à base de $\mu$P en modules fonctionnels, il faut prendre le risque de la standardisation. En effet la décomposition peut être limitée à un certain nombre de boitiers existants ; alors elle sera rapidement dépassée vu la rapide évolution qui existe dans le domaine : on dispose actuellement de cartes microordinateurs.

Heureusement, le besoin de cette décomposition existe et n'est pas prêt de disparaître du fait de l'apparition des systèmes complets sur une pastille (ou puce) ; car cette évolution technologique ne change en rien des concepts de base de la machine de Von Neumann.

Après avoir présenté les décompositions possibles de $SB_{\mu}P$'s, nous décrivons succinctement chacun des modules qui peuvent en faire partie en ne retenant que les caractéristiques principales de ces derniers.

Ensuite nous proposons une façon de regrouper ces caractéristiques, ce qui permettra d'exploiter facilement une base de donnée où sont décrits ces modules.

Enfin, nous donnons 4 tableaux (voir Annexe 3) où sont répertoriés quelques uns des modules actuellement disponibles, pour montrer l'étendue du choix, [BUR 79].

## 5.1 DESCRIPTION FONCTIONNELLE DES MODULES MATERIELS D'UN $SB_\mu P$

Tous les microordinateurs sont réalisés à partir de blocs fonctionnels de base semblabes ; alors que l'organisation et la réalisation interne de ces blocs diffèrent considérablement d'un constructeur à l'autre.

Ces blocs, couramment appelés boîtiers sont implantés autour du bus. C'est lui qui permet aux différents éléments du $\mu\theta$ de communiquer entre eux ; essentiellement entre ceux des boîtiers qui jouent les rôles de maîtres, et qui contrôlent ainsi ceux qui jouent les rôles d'esclaves.

Dans le microordinateur, un seul boîtier est maître à la fois ; ne serait-ce que pendant une courte durée. Peuvent jouer ce rôle :

1. Le contrôleur d'accès direct à la mémoire (ADM), qui accélère le mouvement des données entre les éléments du $SB_\mu P$.
2. Le microprocesseur : il manipule les données d'une façon définie par le programme.

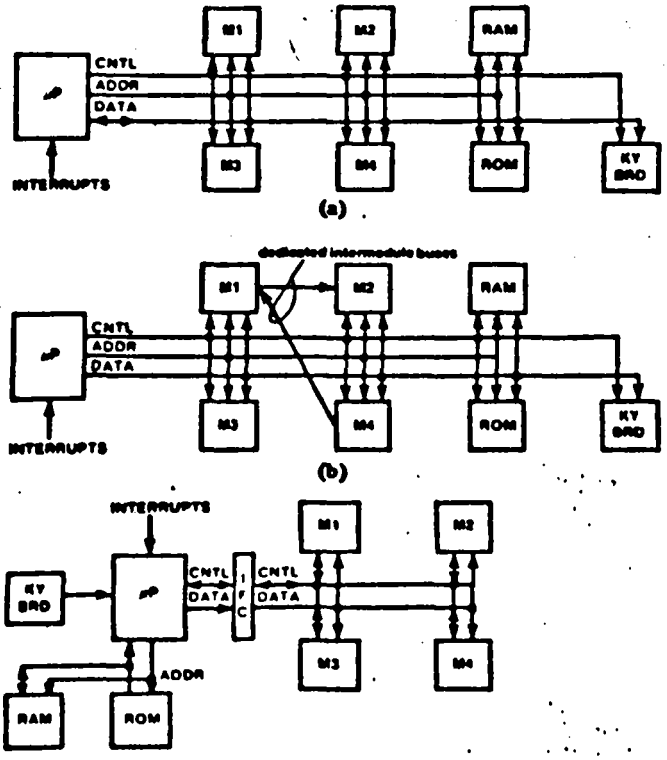
Sont confinés aux rôles d'esclaves :

1. Les mémoires mortes où sont conservés les programmes (ROM).
2. Les mémoires vives : lieux de prédilection des données variables et des résultats (RAM).
3. Les disponibles d'entrée-sortie (E/S), qui réalisent la connexion du $\mu P$ aux périphériques :
   contrôleurs d'unités de disque souple
   contrôleurs de tubes à écran cathodique (CRT)
   etc.

Un traitement effectué par un système à base-de-$\mu P$ peut être appréhendé comme s'étant réalisé avec un échange (continu) d'informations entre les différents modules couplés au bus. Les modules peuvent être considérés comme :

- périphérique (une certaine autonomie)
- esclaves
- ou maîtres d'égale importance avec le $\mu P$.

Suivant les modules à associer au $\mu$P, on choisira l'une des 3 architectures de bus de la figure suivante. Différentes configurations existent et sont décrites dans [RAN 78]. Nous ne les reprendrons pas ici.




Dans la configuration (a), un module $M_i$ (ou composant) est considéré comme un périphérique. Le bus de contrôle et d'adresse est unidirectionnel le bus de données est bidirectionnel. Toutes les communications sont prises en charge par le $\mu$P. Cette configuration est l'une des plus simples à implémenter. Si les huit bits de la donnée ne suffisent pas, les données peuvent être bufferisées et envoyées une à la fois. L'inconvénient de cette configuration est de limiter le débit des informations. Un microprocesseur

lent sera un goulot d'étranglement lorsqu'un nombre important de données devra passer d'un composant $M_i$ à un autre $M_j$. La configuration (b) en est une solution.

Pour la configuration (c), un composant $M_i$ est considéré comme un esclave. Le $\mu P$ traite les données et les oriente vers leurs destinataires, mais aussi il donne des ordres aux esclaves et les contrôle. Cette configuration convient aux $SB_\mu P$ complexes où un grand nombre d'intercommunications entre composants est nécessaire, ainsi qu'un transfert rapide de données. Les esclaves sont doués d'une certaine intelligence pour exécuter les ordres.

Dans la configuration (c) le microprocesseur est surtout un contrôleur il ne "voit" pas les données qui transitent entre les autres composants par le bus.

## 5.2 LES MODULES (MATERIELS) DU $SB_\mu P$

Avant de donner les caractéristiques des modules, nous allons brièvement décrire l'élément qui réalise les liens entre eux :

### 5.2.1 Le bus

On décompose souvent le bus selon la structure suivante :

a) Le bus d'adresse

Il est unidirectionnel et par lui transitent des adresses, pour la

sélection - des positions mémoires

des interfaces de périphériques.

Il peut comporter 12, 14 ou 16 lignes.

b) Le bus de données

Il est bidirectionnel et par lui transitent les instructions et les données, entre les différents modules. Le nombre de lignes le constituant est lié à la longueur de mot du $\mu$P (4, 8, 12 ou 16 bits).

Un bus banalisé est celui qui joue les rôles a) et b).

c) Le bus de contrôle

Son rôle est le transport des différents signaux de commande.

### 5.2.2 Les Maîtres

Les travaux de plusieurs maîtres peuvent s'entrelacer et un maître peut louer ou donner le contrôle du bus à un autre maître. Dans chaque échange d'informations, le maître sélectionne, à travers le bus, l'esclave destinataire.

Nous distinguons deux types de boîtiers pouvant jouer le rôle de maître :

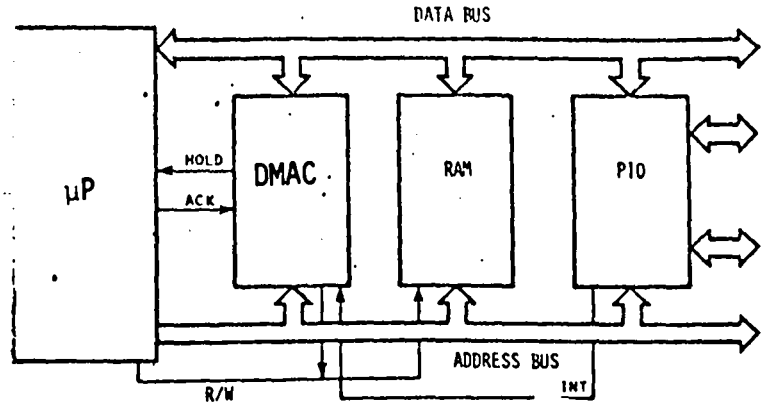
- les circuits d'adressage direct de la mémoire
- le $\mu$P lui-même.

Plusieurs ouvrages de micro-informatique décrivent ces circuits. Et souvent il est préférable de se documenter auprès du constructeur.

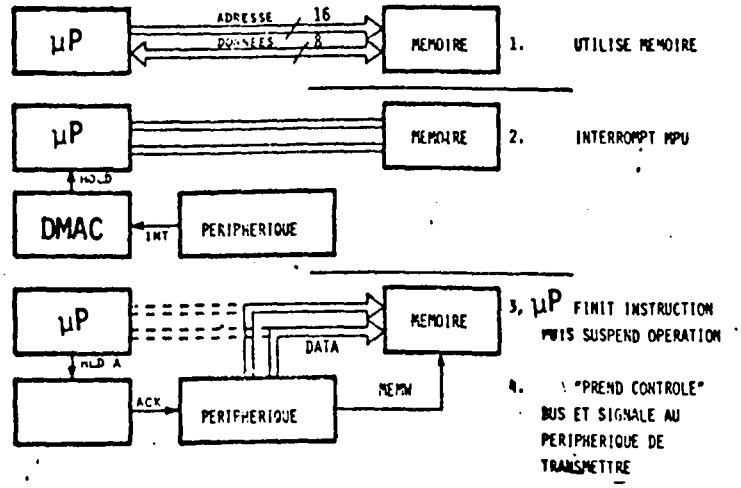
#### 5.2.2.1 *Accès direct à la mémoire*

*Définition :*

Lorsque l'on a de grandes quantités de données à transférer en mémoire à cadence élevée (sur bande magnétique ou sur disque souple par exemple), il peut être trop long de passer par l'intermédiaire du $\mu$P. Pour cette raison il existe une procédure spéciale, permettant de "court-circuiter" le $\mu$P au cours de ces transferts : c'est l'accès direct à la mémoire.




Utilisation d'un DMAC




: Opération du DMAC

Les interruptions garantissent la réponse la plus rapide possible pour un périphérique. Toutefois le service du périphérique est logiciel (d'où la lenteur possible pour les processus qui mettent en jeu des transferts mémoires rapides). On remplace le logiciel par le matériel. La routine logicielle effectuant le transfert entre la mémoire et le périphérique, est remplacée par un processeur (matériel) spécialisé : le DMAC ou contrôleur d'accès direct à la mémoire.

Le DMAC est donc conçu pour effectuer des transferts rapides de blocs (chaînes de caractères) directement entre la mémoire et le périphérique.

Un DMAC est utilisé dans les systèmes où :
l'utilisation d'une unité de disque souple ou
d'un écran cathodique est nécessaire.

Il existe trois types d'accès direct à la mémoire :
- l'accès direct avec arrêt du $\mu$P.
- l'accès direct avec ralentissement du $\mu$P ou vol de cycle.
- l'accès direct par multiplexage.

Ces types sont parfaitement décrits dans les livres de micro-informatique, comme [ZAK 77] ; nous ne nous y attardons donc pas.

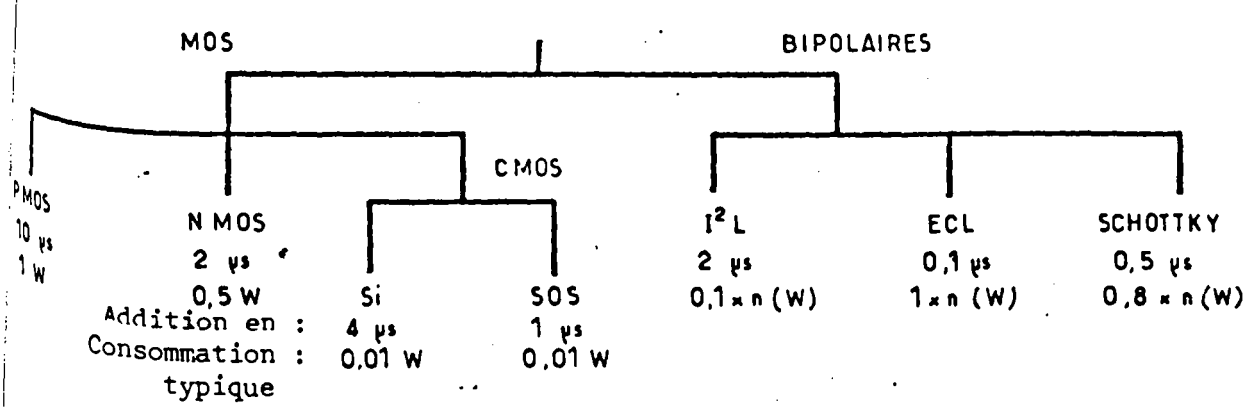
Les variables que nous avons retenues comme pouvant caractériser un boîtier d'accès direct à la mémoire sont :
- le temps de réponse aux demandes d'accès direct. Il dépend du microprocesseur et peut intervenir dans sa sélection.
- le débit maximum obtenu au cours d'une opération d'A.D.M. C'est le ratio maximum de transfert,en byle/cycle.
- le temps de cycle mémoire dont dépend le débit maximum.
- la fréquence de l'horloge, enfin.

### 5.2.2.2 *Le microprocesseur*

Le $\mu$P est capable de réaliser un certain nombre d'opérations, au rythme des impulsions d'une horloge et sous le contrôle des programmes lus en mémoires.

C'est un module multifonction qui allie aux propriétés combinatoires et séquentielles, la propriété d'être programmable. Il ne peut effectuer qu'une opération à la fois ; citons :
- les opérations arithmétiques et logiques
- le contrôle et l'utilisation des mémoires
- le contrôle des interfaces (d'entrées / sorties)

Il peut être décrit à partir des caractéristiques suivantes :
- la technologie, qui permet de juger de la rapidité de calcul et de la consommation.




Technologies pour microprocesseurs

ce qui permet de connaitre sa puissance à l'aide du cycle de base.

- la longueur de mot ; à laquelle nous reviendrons au chapitre sur la sélection de $\mu$P's.
- le constructeur, ce qui permet entre autre de la situer dans sa famille.

etc.

### 5.2.3 Les Esclaves

Dans les esclaves nous mettrons :

1. Les mémoires
2. et les dispositifs d'entrée-sortie

#### 5.2.3.1 *Les mémoires*

Il en existe une grande variété utilisable dans les SB$\mu$P, comme le montre le tableau de la page suivante. Nous ne décrirons que le premier des 3 groupes : celui des mémoires à accès aléatoires.

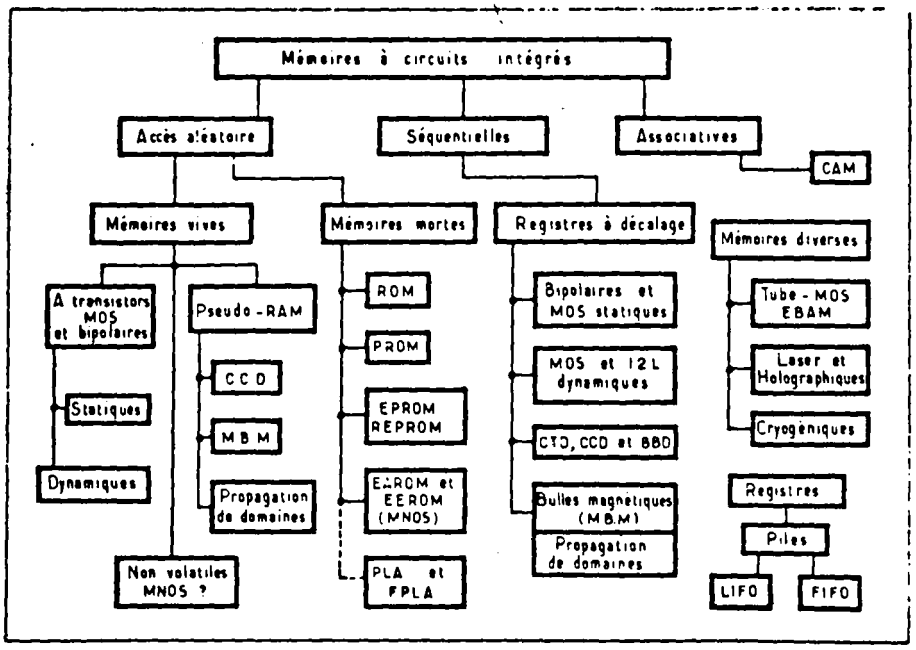



**Tableau général des mémoires en circuits intégrés (ou liées à l'intégration).**

a1) Les mémoires mortes (ROMs) ou (mémoire de programme)

Ce terme de mémoire morte est appliqué à une vaste tamme de "produits" dans lesquels sont stockés deux états d'informations pouvant être retrouvés plus tard : le "produit" étant habituellement un circuit intégré contenant une matrice de cellules adressables, chaque cellule ayant été placée de façon définitive dans un état binaire 0 ou 1.

La ROM sert à garder :

- le programme (des instructions)
- les données constantes (ex : tables numériques de références).

Chaque ROM (boîtier) a une capacité de 1024 octets et a une organisation par mots de un ou plusieurs bits, la lecture se faisant mot par mot.

Les variables caractéristiques :

- capacité (nombre de boîtiers)
- temps de cycle ou le temps d'accès (vitesse)
- organisation [+ des mots (nombre de bits)] our architecture
- adaptabilité et souplesse d'emploi (ROM, RPOM)
- enfin le prix.

a2) Les mémoires vives (RAM(S))

La mémoire vive est utilisée pour stocker des informations variables dans S B µ P . Ainsi le µP, sous le contrôle du programme, peut à volonté lire ou changer le contenu d'une position de la mémoire vive.

La plupart des RAM sont volatiles.

Il existe deux sortes de RAM :

- Les RAM statiques.
- Les RAM dynamiques où l'information est stockée sous forme de charge électrique dans la capacité de poste d'un transistor MOS. Comme ces condensateurs ne sont pas parfaits, les charges s'écoulent par des fuites et l'information est perdue si la charge n'est pas régénérée (ou rafraîchie) périodiquement.

Les RAMS permettent :

. l'enregistrement des informations variables (données) mot par mot.

. La mémoirisation globale des mots inscrits.

. La lecture non destructive des mots (mot par mot).

**RAM semi-dynamiques et dynamiques 1 K comparées** [LIL 77]

| CARACTÉRISTIQUES (1) | AMS 6002 | INTEL 1103 | MOSTEK MK 4006 | NATIONAL MM 5260 | INTEL 1103-1 | FAIRCHILD 3534 | ELECTR... |
|---|---|---|---|---|---|---|---|
| Organisation | 1024 x 1 | 1024 x 1 | 1024 x 1 | 1024 x 1 | 1024 x 1 | 1024 x 1 | 1024 |
| Fréquence de rafraîchissement (ms) | 2 | 2 | 2 | 2 | 1 | 2 | 1 |
| Température de service (°C) | 0-70 | 0-70 | 0-70 | 0-70 | 0-55 | 0-70 | 0-7 |
| Alimentation | +7 V, +20 V +22,5 | +16 V, +20 V | +5 V, -12 V | +5 V, -12 V | +19 V, +22 V | +16 V, +20 V | ±15 V ±12 V |
| Polarisation du substrat | Oui | Oui | Non | Non | Oui | Oui | (1) |
| Accès (ns) | 150 | 300 | 400 | 350 | 150 | 300 | 150 |
| Cycle lecture (ns) | 250 | 480 | 400 | 450 | 340 | 480 | 150 250 |
| Cycle écriture (ns) | 250 | 580 | 650 | 600 | 340 | 580 | 250 400 |
| Synchronisation critique | Non | | Non | Non | | Non | |
| Consommation au repos (µW/bit) | 2 | 60 | 35 | 75 | 76 | 4 | |
| Consommation dynamique (mW) | 180 | 400 | 450 | 400 | 500 | 200 | |
| Compatibilité TTL | Non | Non | Oui | Oui | Non | Non | |
| Technologie | PMOS Gate Al haut seuil | PMOS Gate Si bas seuil | PMOS $I^2$ (1) bas seuil | PMOS Gate Si bas seuil | PMOS Gate Si bas seuil | PMOS Gate Si bas seuil | |
| Transistors par cellule | 4 | 3 | 3 | 3 | 3 | 4 | |
| Nombre d'accès au boîtier | 22 | 18 | 16 | 16 | 18 | 18 | |

(2) - $I^2$ = implantation ionique

(1) On retiendra d'abord celles qui sont soulignées.

### 5.2.3.2 Les dispositifs E/S ou équipements d'E/S ou interfaces

Comme un $\mu$P ne fonctionne pas isolément, ses échanges avec un périphérique nécessitent des outils composés de deux parties :

1) une partie matérielle comprenant les circuits d'interface proprement dits ; on en distingue 2 types différents :

a) les circuits d'interface

b) les contrôleurs de périphérique.

2) une partie logicielle comportant le programme de gestion du périphérique

a.3.1. Partie matérielle

*a.3.1.1 Le circuit d'interface*

C'est l'intermédiaire direct entre le $\mu$P et l'extérieur ; le $\mu$P le sollicite pour toute instructions d'e/s. Il comporte :

- un registre tampon d'interface où sont chargées provisoirement les données tranférées.
- Une porte e/s donnant accès au tampon et validée par les commandes de lecture/écriture du µP.

C'est à ce niveau qu'il faut déterminer le mode de transmission des informations (série ou parallèle).

Pour les sytèmes importants, on dispose de périphériques performants d'où nécessité d'interfaces programmables PPI, PCI, PIA, ACIA, SSDS, etc .

En général le rôle d'une interface est :

. De transmettre une donnée, en effectuant éventuellement l'adaptation nécessaire :

comptabilité MOS/TTC ;

tranfert série ou parallèle ;

etc.

. D'obéir aux signaux de contrôle envoyés soit par le $\mu$P, soit par le périphérique.

Deux situations sont possibles :

1) Des adaptateurs d'interfaces permettent aux périphériques de réaliser le contrôle des routines (gestionnaires) le servant.

2) Les modules d'interface sont traités comme des positions mémoire (dans certaines famille de $\mu$P's), avec l'avantage de pouvoir appliquer toutes les instructions et méthodes d'adressages relatives aux mémoires (d'où un contrôle direct du périphériques).

En résumé : l'interface

- traite des requêtes non synchronisées
- sélectionne des demandes des périphériques et
- expédie, ou traite les interruptions.

### *a.3.1.2 Le contrôleur de périphérique*

Cest un ensemble de circuits logiques, externes au $\mu$P, adaptés à un périphérique donné pour en assurer le bon fonctionnement.

Exemple : Coupleur de téléimprimeur :

Un caractère frappé sur clavier est codé en mode série, c'est une chaîne de 8 bits à coder en ASCII ou en EBCDIC. C'est le contrôleur qui assure cette opération, met en forme les signaux générés. Il existe des périphériques intelligents où le contrôleur est lui même un micro-ordinateur.

## a.3.2 Partie Logicielle

### Le gestionnaire d'unité

Une fois l'interface matérielle réalisée avec le $\mu$P, il faut lui fournir un programme lui indiquant comment assurer les échanges avec le périphérique.

Ce programme doit contrôleur les fonctions suivantes ;

- lors d'une opération d'entrée ; la bonne interprétation des signaux qui peuvent être sous forme série ou parallèle, générés de façons diverses (électrique, électromécanique ...)
- lors d'une opération de sortie : la mise en forme convenable par le $\mu$P, de l'information suivant la nature du périphérique

A chaque périphérique correspond un gestionnaire spécifique, en mémoire sous forme de sous programme, auquel le $\mu$P se branche lors d'une e/s [ZAKS 77].

Sauf cas très exceptionnel, le microprocesseur ne pourra travailler seul. Il faudra lui adjoindre un certain nombre de boitiers réalisant des fonctions particulières.

Outre les boitiers maintenant classiques :

- . mémoires,
- . horloges multiphases,
- . registres tampons
- . interfaces d'entrée-sortie ;

on trouve

- . des contrôleurs d'accès direct à la mémoire
- . des contrôleurs d'unités à disques souples
- . des compteurs d'intervalles de temps
- . des contrôleurs de tubes à écran cathodique
- . des coupleurs de claviers et d'afficheurs
- ... etc.

Ces boitiers ont la particularité d'être programmables. En ce sens que les ordres codés issus du processeur central leur imposent la modalité de fonctionnement, convenable parmi toutes celles qui sont autorisées.

La pleine programmabilité est atteinte par les $\mu$P's esclaves, chargés de travaux particuliers, tels que gestion des périphériques.

Exemples : les 8041 et 8741 d'Intel.

EXEMPLES DE CIRCUITS D'INTERFACE

- CIRCUITS PROGRAMMABLES
  - SERIE
    - 6850 MOTOROLA
    - 8251 INTEL
    - 6854 MOTOROLA
    - 6852 MOTOROLA
  - PARALLELE
    - E/S Seulement
      - 6821 MOTOROLA (PIA)
      - 8255 INTEL (PIO)
      - PIO/Z-80
      - ⋮
    - E/S + d'autres fonctions
      - 8155 E/S + RAM + TEMPORISATEUR
      - 8355 E/S + ROM + TEMPORISATEUR
      - 8755 E/S + PROM + TEMPORISATEUR
      - 6846 E/S + ROM + TIMER
- CIRCUITS PROGRAMMABLES ORIENTES
  - *TIMERS, TEMPORISATEURS
  - *CONTROLEURS DMA
  - *CONTROLEUR D'INTERRUPTIONS
  - *COUPLEUR DE DISQUE SOUPEL
  - *COUPLEUR D'ECRAN CATODIQUE
- CIRCUITS NON PROGRAMMABLES
  - INTEL 8205 — one of 8 decoder → décodeur
  - INTEL 8214 — unité de contrôle des priorités des interruptions
  - ⋮
  - AM 9511 — Processeur arithmétique AMD
    - + - x -
    - trigono 16 bits
    - √ 32 bits
    - exponentielle

*Configuration des broches fixée par le constructeur dès le départ.*

## 5.3 LES CONCEPTS DIRECTEURS POUR LA CONSTRUCTION DE LA BASE DE DONNEES

Nous avons passé en revue les différents composants d'un SBµP. Nous allons maintenant reprendre les caractéristiques et donner les concepts auxquelles elles correspondent. L'objectif étant de guider le concepteur lors de l'exploitation du fichier CATALOGUE où sont décirts les différents composants d'un SBµp.

Nous divisons l'ensemble de l'équipement présent sur le marché en deux groupes.

a) Circuits élémentaires de base : RAM, ROM, µP (éventuellement horloge, RAM, ROM, etc incorporeer).
Exemple : 8080, 6800, 8048, Z 8000, F8, etc.

b) Carters assemblés (ou en kits) de base.
Exemple : SBC 80/10, 80/20, 80/30 d'INTEL, MICROMODULES de MOTOROLA, MD de MOSTEK, etc. Il existe des modules d'expension d'E/S, mémoire, etc.

### 5.3.1 Caractéristiques à retenir

Nous divisons pour ceci les caractéristiques en

1. Spécifications fonctionnelles
2. Spécifications électriques et mécaniques

et nous donnons un EXEMPLE :

#### 5.3.1.1 Spécifications fonctionnelles

. Identification : SBC 80/20

Le µP utilisé est le 8080 A ;
la longueur de mot est de 8 bits pour les données ;
et celles des instructions est 8, 12 ou 24 bits.

. Les mémoires disponibles sur les cartes :

Mémoire vive (RAM) : de type statique
Capacité 2KØ avec des extensions possibles
adressage : de zéro à 64 K

Mémoire morte (RCM / EPROM)
Capacité 8KØ
adressage de 0 à OFFF

Les entrées/sorties

Pour les E/S parallèles on dispose de 48 lignes programmables.

Pour les E/S série :

La carte contient un USART, avec une interface compatible RS232.

Il est possible de travailler en 2 modes.

1) Le mode synchrone :

On dispose alors de 5 ou 8 bits de synchronisation ; celle-ci peut être interne ou externe, et il est possible de l'insérer de manière automatique.

2) Le mode asynchrone

Là aussi on a 5 ou 8 bits de synchronisation. En plus il est possible de générer un caractère de "BREAK" ; et on a 1, 1 1/2 ou 2 bits de stop.

Pour les deux modes, les vitesses de transferts différentes existent et dépendent de la fréquence de l'horloge.

Les interruptions peuvent venir de 26 sources différentes ; on peut les embrocher les vectoriser ou les traiter par scrutation.

Huit niveaux d'interruption.

Les compteurs d'évènements pour générer des intervalles de temps précis, sous le contrôle du logiciel, sont au nombre de deux.

## 5.3.1.2 Spécifications électriques et mécaniques

Toutes les interfaces sont compatibles TTL

Sources d'alimentation - 5 V ; - 12 V ; + 5 V ; 12 volts

Fréquence de l'horloge 2,1504 MHz ± 0,1 %

Cycle de base de l'instruction : 1,86 µs

Plage de température ambiante acceptable : de -0°C à 55°C

Caractéristiques physiques : longueur 30,48 cm
largeur 17,15 cm
Epaisseur 1,26 cm
Poids 397,6 grs

Des cartes assemblées permettent des extensions
de mémoire jusqu'à 64 K ;
et des E/S ;
ainsi que pour le contrôleur d'unités de disquette
et les alimentations.

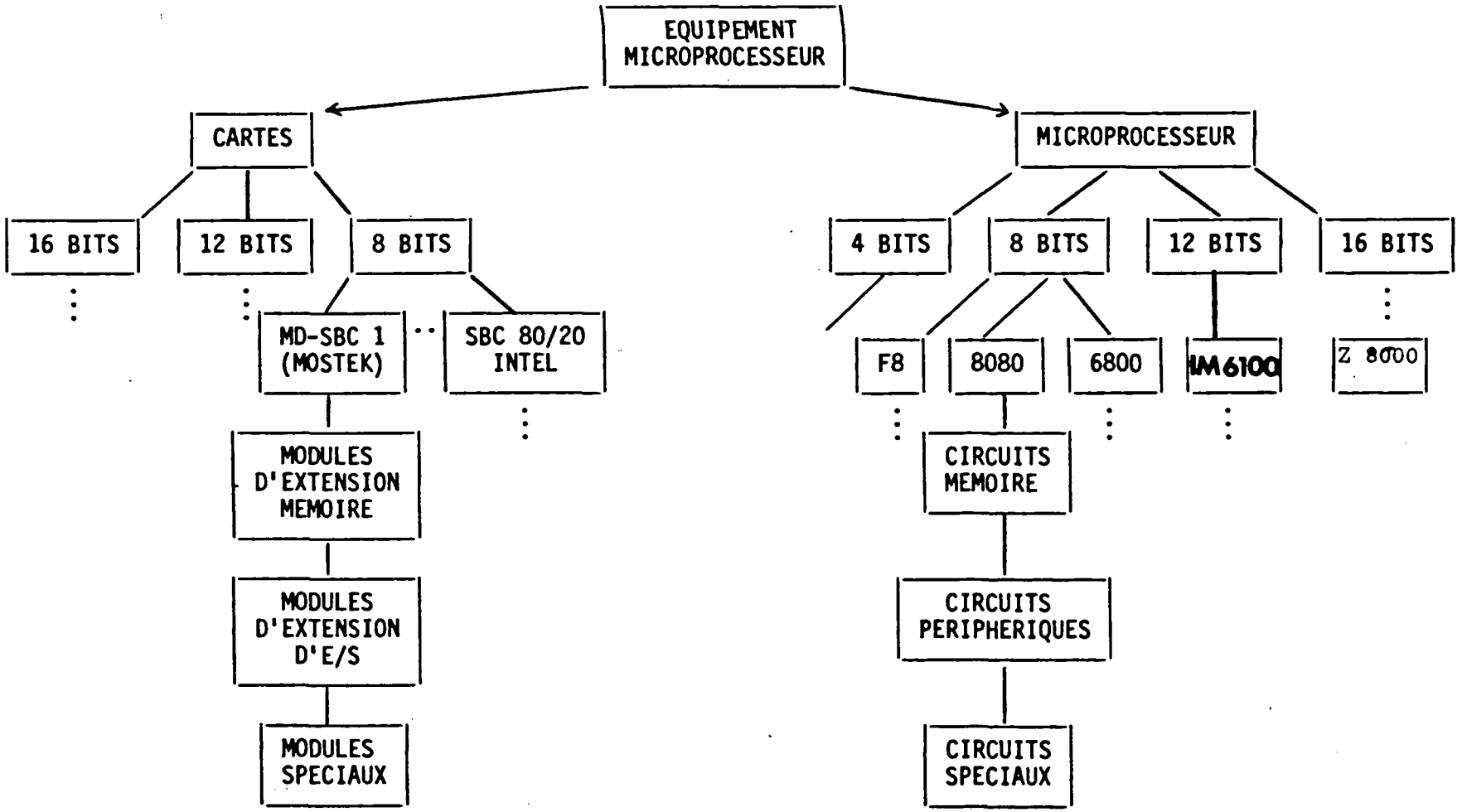
Cette description à partir des 3 concepts fait apparaitre très simplement les caractéristiques communes de cartes ayant la même fonction.

### 5.3.2 Approche de description de la base

La demande suivie pour la décomposition des cartes en modules de base et les éventuelles cartes d'extension, est la même que celle à suivre pour les circuits élémentaires de base :

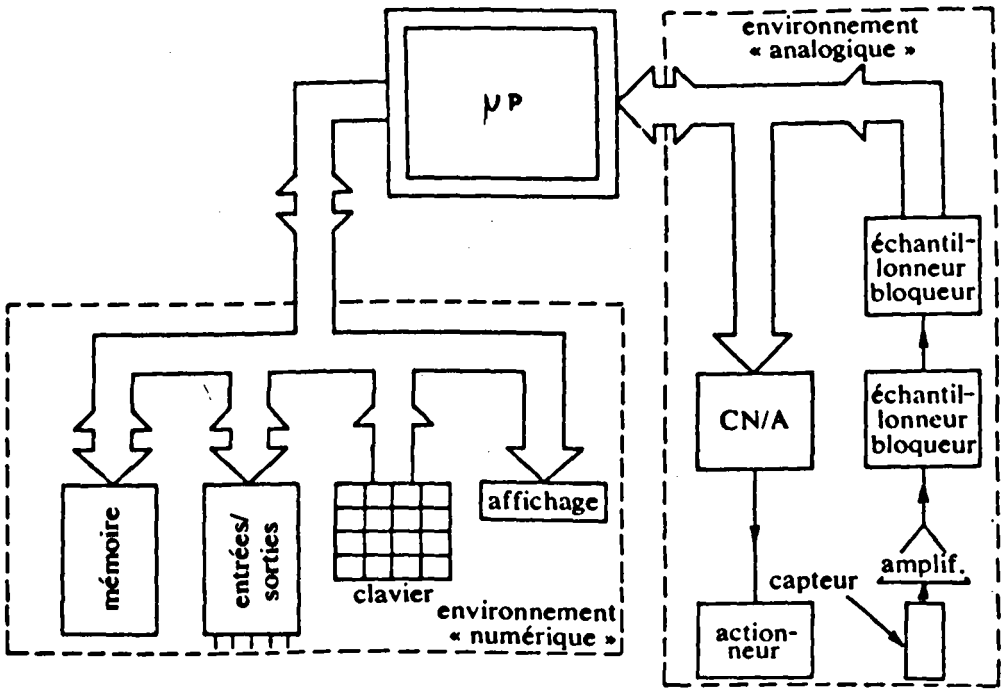
1. Description du microprocesseur
2. Description des circuits associés
   - circuits périphériques : décodeurs, ports d'E/S, unités de contrôle d'interruption, générateurs d'horloger, gestionnaire de bus, coupleur d'E/S, etc ...
   - circuits mémoire : boitier RAM, ROM, PROM, EPROM, etc ...
   - circuits spéciaux : registres à décalage, mémorier FIFO, encodeur clavier, générateur de caractères, etc ... [CUS 79].

Cela permettra à COSMIC, lors des phases de : choix du matériel microprocesseur et programmation des circuits d'interface de puiser la description de cette base des données.


EQUIPEMENT MICROPROCESSEUR
CARTES
MICROPROCESSEUR
16 BITS
12 BITS
8 BITS
4 BITS
8 BITS
12 BITS
16 BITS
MD-SBC 1 (MOSTEK)
SBC 80/20 INTEL
F8
8080
6800
IM6100
Z 8000
MODULES D'EXTENSION MEMOIRE
CIRCUITS MEMOIRE
MODULES D'EXTENSION D'E/S
CIRCUITS PERIPHERIQUES
MODULES SPECIAUX
CIRCUITS SPECIAUX

# CONCLUSION

Le microprocesseur, associé à des mémoires vives, mortes et à des circuits d'interface, constitue un micro-ordinateur. Un micro-ordinateur peut se présenter sous deux formes, qui sont autant des choix pour le concepteur. Il peut partir d'un kit ou ensemble de boitiers à assembler ; d'une carte micro-ordinateur, dont certaines sont dédiées à un type d'application particulier, ou même un micro-ordinateur complet et utilisable sans ajouts. L'objectif étant de proposer une configuration de SBµP adéquate à l'utilisateur [GLA 73].




Si l'interfaçage avec un environnement analogique fait toujours appel aux mêmes éléments constitutifs, il n'en est pas de même avec les interfaçages numériques : autant de systèmes numériques à coupler à un processeur, autant d'interfaces spécialisées aux caractéristiques bien particulières. [ROB 79].

Nous avons vu qu'il y a une diversité de choix ; d'où la nécessité d'examiner certaines des caractéristiques, de disposer de critères de sélection pour sortir de ce labyrinthe.

Le cahier des charges n'a d'intérêt que s'il permet d'établir le chemin pour construire les SBµP's, et ces derniers sont issus de l'assemblage des éléments du deuxième ensemble de connaissances.

Dans tous les cas, le rôle de COSMIC est d'assister le concepteur. Pour le choix de la configuration matérielle, les algorithmes de choix que nous développons dans la suite exigent la disponibilité des caractéristiques des composants de base (ou éléments). Caractéristiques qu'on retrouve en partie dans les 4 tableaux en Annexe 3. Ce deuxième ensemble de connaissances sera une base de données (technologique), aisément consultable : le CATALOGUE.

# BIBLIOGRAPHIE

[BUR 79] D. BURSKY
"*1979 µC data Manual*" EDN N° 5 March 1 1979, p 45-80.

[CUS 79] R. CUSHMAN
"*Are single-chip microcomputers the universal logic of the 1980 s ?*"
EDN 5 Jan. 1979

[GLA 73] GLADSTONE
"*Designing system around* µP". Electronics p 104, 11 oct. 73

[LIL 77] H. LILEN
"*Mémoires intégrées*",Editions Radio 1977,

[PLE 79] D. PLEMENOS
"*Etude et Réalisation d'un système de CAO appliqué à la micro-informatique*". Thèse Lille 79.

[RAN 78] C. RANDLE et N. KERTH
"µP *in instrumentation*", Proceedings of the IEEE Fev, 78, Vol 66
N° 2 ; p 175.

[ROB 79] M. ROBIN, Th. MAURIN
"*Interfaçage des microprocesseurs*" Dunod Technique 1979

[ZAK 77] A. LESEA et R. ZAKS
"*Microprocesseur interfacing Techniques*" SYBEX 77,

# CHAPITRE VI : SÉLECTION DES CONSTITUANTS DE LA CONFIGURATION RÉPONDANT AUX BESOINS DE L'UTILISATEUR

INTRODUCTION

6.1 HIERARCHISATION SIMPLE DES CRITERES DE SELECTION D'UN SYSTEME

- 6.1.1 Les critères les plus importants
- 6.1.2 Les critères de deuxième niveau d'importance
- 6.1.3 Les autres critères

6.2 ETUDE ET CRITIQUE DES ALGORITHMES DE SELECTION DE MICRO-PROCESSEURS EXISTANTS

- 6.2.1 Algorithme de BAYON
- 6.2.2 Algorithme de TABAK

6.3 PROPOSITION D'UNE DEMARCHE SIMPLE DE CHOIX

6.4 CHOIX DES MEMOIRES

6.5 LES STANDARDS

- 6.5.1 Les bus standards
- 6.5.2 Les interfaces
- 6.5.3 Les circuits dédiés
- 6.5.4 Ou en est la standardisation du logiciel

CONCLUSION

BIBLIOGRAPHIE

# INTRODUCTION

Il est nécessaire, pour le concepteur, d'avoir un moyen lui permettant de choisir les circuits qui vont constituer son SBµP.

Nous pensons qu'il n'est pas possible de prendre en compte tous les critères pouvant orienter le choix vers telle ou autre famille de microprocesseurs. Car il est difficile de quantifier des considérations du genre :

- expérience de l'utilisateur dans l'une des familles ;
- nombre de secondes sources du constructeur*

Bien sûr ce genre de considérations n'est pas négligeable, mais nous avons choisi de ne traiter que de l'aspect technique de la conception.

Nous avons souligné au chapitre le fait que le concepteur pouvait choisir :

- soit de construire son système en partant du µP
- soit utiliser des cartes assemblées.

Certains critères de choix sont déductibles du cahier des charges, donc déterminants. Mais nous parlons aussi des autres critères : ceux que seul le concepteur peut appréhender.

Des algorithmes ou méthodes de choix ont été proposé dans la littérature. Après avoir brièvement exposé deux d'entre eux, nous en faisons une critique.

Ce qui nous permet de proposer une démarche simplifiée, pour atteindre le même but : celui de choisir le microprocesseur adéquat.

* Une seconde source est une entreprise qui revend, fabrique ou utilise dans ses (propres) produits des circuits venant d'un constructeur.

## 6.1 LES CRITERES LES PLUS IMPORTANTS

Dans le chapitre 5 précédant nous avions indiqué l'existence de trois démarches possibles pour construire le SBμP à partir soit d'un kit pour assembler les différents boîtiers, soit d'une carte assemblée. La troisième démarche étant d'acheter un micro-ordinateur. Pour diriger le choix vers l'une de ces trois solutions, le critère utilisé est : le facteur de répétitivité de l'application.

Le coût, fonction très complexe à évaluer, dépend lui aussi du facteur de répétitivité. Quoique déterminant dans bien des cas de conception, nous ne le prenons pas directement en compte car c'est une considération économique par excellence.

Il peut s'exprimer de la façon suivante :

$C_A = C_P + C_P$ (A comme application) ;

où $C_D$ est le coût de développement, et $C_P$ celui de la production. $C_P$ est le produit du nombre de système à produire par le coût d'un système.

### 6.1.1 La longueur de mot

La longueur de mot constitue une des principales caractéristiques d'un micro-ordinateur. Elle indique le nombre de bits qui peuvent être lus ou écrits dans la mémoire à chaque cycle. Plus la longueur de mot est grande, plus la quantité d'informations manipulée à chaque cycle est importante.

Pourquoi choisir une longueur de mot plutôt qu'une autre ?

Parce qu'elle doit être adaptée aux données à traiter.

1° Si l'on travaille en décimal codé binaire (BCD), puisqu'un digit décimal est codé par un mot de 4 bits, on peut utiliser des mots de 4 bits, les microprocesseurs 4 bits étants largement abandonnées, on utilisera des µP's 8 bits.

2° Si l'on travaille en code ASCII avec un téléimprimeur par exemple celui-ci étant codé sur 7 bits il faudra choisir un µP de 8 bits.

3° Dans l'industrie, des conversions analogiques numériques de grandeurs pourront se faire sur 8 bits, mais 12 bits offrirons une bien meilleure précision. Ex. : Système d'acquisition de données.

4° Enfin les problèmes généraux de gestion et de calculs scientifiques devront faire appel aux 16 bits et éventuellement plus (32).

Nous voyons ainsi que ce critère caractérise la puissance du jeu d'instruction car généralement, dans le cas des µP, cette dernière est un multiple de la longueur de mot. L'inconvénient d'une telle organisation étant de demander plusieurs cycles pour lire une instruction. D'où des temps d'exécution relativement lents par rapport aux mini-ordinateurs.

### 6.1.2 Le facteur de répétitivité de l'application

C'est le nombre de systèmes identiques qu'il faut réaliser pour un utilisateur donné. Plusieurs auteurs distinguent ainsi, à partir de leur expérience :
des applications de petite série : moins de 50 systèmes et
des applications de grande série : plus de 100 exemplaires.

On parle aussi de taille de la série.

6.1.2.1 Pour des applications de petite série, on utilise des cartes ou des systèmes bâtis à partir des cartes fournies par le constructeur. On peut ainsi limiter les coûts de développement (en particulier ceux de la programmation), qui sont ici plus importants devant ceux de la production.

6.1.2.2 Pour des applications de grande série, ce sont les coûts de production qui sont prépondérants. Leurs réduction implique une optimisation au niveau du matériel. Il faut donc bâtir une électronique adaptée, où on utilisera suivant les performances souhaitée soit des µP bipolaires en tranches, soit des µP MOS monolithiques.

A ce niveau, la programmation en langage d'assemblage peut être avantageuse.

Les différents boîtiers qui constitueront le SBµP peuvent quelquefois être connectables au µP sans logique additionnelle. Ceci n'est possible que s'il sont de la même famille. Lorsqu'il y a exogamie, les spécifications de bus différentes peuvent nécessiter une circuiterie de couplage (plus) importante. Il est alors plus logique d'associer le facteur de répétitivité de l'application à la notion de famille de µP.

### 6.1.2 Les critères de deuxième niveau

Un cahier des charges n'est jamais présentée sous une forme normalisée. Le concepteur ne dispose donc pas à priori de réponse à une question qu'il pourrait se poser ; il lui faut la déduire à partir d'informations éparses ou, implicites. Heureusement tel n'est pas toujours le cas : certaines réponses sont explicites.

Les critères de deuxième niveau sont ceux qu'il faut déduires. Dans la suite, ils sont notés $D_i$, i = 1 à 6.

Ils dépendent du cahier des charges.

D'autres critères, qui ne dépendant pas directement du problème de l'utilisateur sont appelés secondaires : $S_j$, j = 1, 2, 3,...,n.

Le µP doit pouvoir accéder à ses périphériques en même temps qu'il exécute une tâche ; et de manière générale les E/S sont traitées par interruption, ou par (ADM) accès direct à la mémoire.

$D_1$ La longueur de la pile. Le changement de contexte, lié aux interruptions peut être fait automatiquement, ou par logiciel. Le contexte est stocké dans la pile qui est

- soit sur le boîtier, alors sa taille est limitée, mais l'accès est plus rapide (pile en matériel).
- soit dans la RAM, et c'est l'inverse de la situation précédente (pile programmée).

$D_2$ Le nombre de registres de travail influe sur le temps d'exécution d'une tâche. Plus il y a des registres de travail, et moins la tâche utilise la mémoire. D'où un gain du point de vu rapidité, excepté en interruption si le contexte doit être sauvegardé en entier. On distingue :

le nombre d'accumulateurs,
et le nombre de registres internes.

$D_3$ L'accès direct mémoire. Certains µP ne le possèdent pas, il est alors nécessaire d'ajouter des dispositifs logiques supplémentaires permettant de bloquer le µP ou d'isoler ses bus par des portes 3 états.

Pour ceux qui le possèdent, on peut choisir à l'aide du tableau suivant

| Sous-critères Techniques | Ratio maximum de transfert ADM | Effets sur le µP et l'exécution du programme | Relative complexité du matériel |
|---|---|---|---|
| Arrêt du processeur (Halt) | 1 octet par cycle d'horloge | µP arrêté | la moins |
| Vol de cycles | 1 octet pour 2,5 cycles | MPU actif 1 cycle sur 5 | quelque |
| ADM multiplexés | 1 octet par cycle (*) | Cycle µP ralenti | la plus |

(*) Cependant, le cycle est plus long.

$D_4$ Les possibilités arithmétiques. Un µP peut avoir dans sont répertoire des instructions arithmétiques BCD et éventuellement la multiplication et la division. Si les E/S sont à faire en BCD, les traitements sont grandement facilités [BRI 78] si ces instructions existent. On évite alors des transcodages BCD - binaire puis binaire BCD souvent sources d'erreurs numériques pour des calculs sur des flottants. Les calculs complexes sont plus simples à écrire, moins encombrants et plus rapides si on dispose de la division et de la multiplication.

$D_5$ Le nombres d'interruptions (non cascadables). Il est possible de déduire du cahier des charges : le nombre d'interruptions nécessaires, la fréquence de ces interruptions. Les gains les plus connus des interruptions sont : une réponse rapide à des évènements aléatoires asynchrone dont la nature doit modifier le cours du programme, une possibilité d'utiliser la vitesse du µP pour traiter plusieurs périphériques lents, une possibilité de simplification de circuits et de matériel. Dans les µP's 8 bits, les controleurs d'interruptions peuvent gêner un maximum de 8 interruptions non cascadables. Pour ceux des µP's dont la longueur de mot est 16 bits, on a 16 broches d'interruptions. Lorsque ces limites sont à dépasser (indications du cahier des charges), on peut alors soit faire des interruptions en cascades, soit utiliser des interruptions vectorisées.

### 6.1.3 Les autres critères (troisième niveau)

$S_1$ Les modes d'adressages sont un critère secondaire car les µP possèdent la plupart d'entre eux.

Seuls l'adressage indirect et l'adressage indexé sont à examiner.

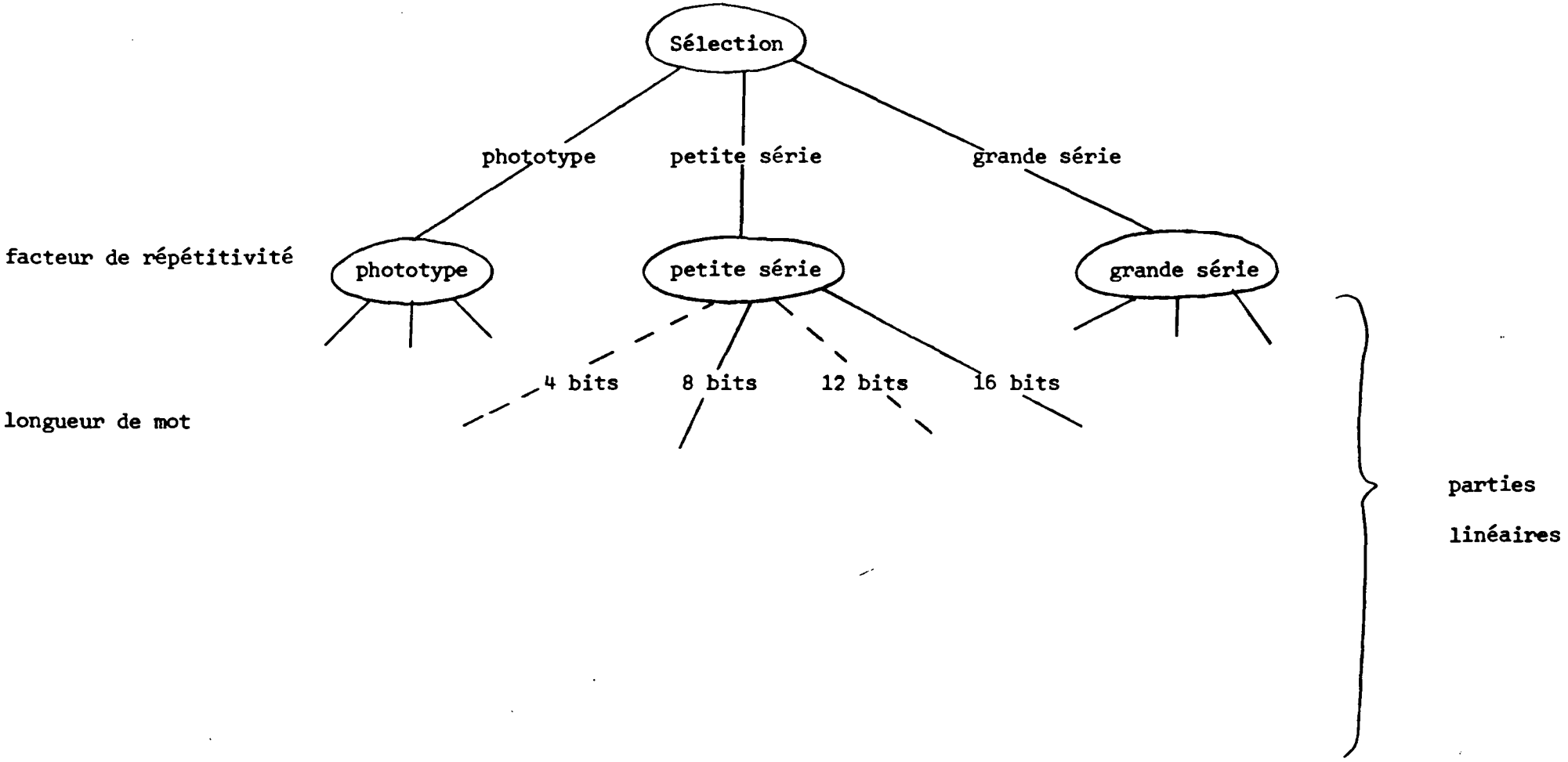
$S_2$ La capacité d'adressage . Elle correspond à la taille mémoire maximale adressable par le µP. Les µP en tranches permettent de l'adapter au problème à traiter ; et s'il faut l'examiner attentivement pour les µP's en un seul boitier ou les µP's 4 bits, elle est de 64 K pour la plupart des µP's

8 bits ; et au delà de 64 K pour les 16 bits. L'idéal étant d'adapter la capacité d'adressage à la taille mémoire requise.

$S_3$ L'horloge. Elle est soit intégrée au processeur, soit fournie sous forme de boîtier de la famille. Sinon il faut la réaliser soit-même, et ce dernier cas est peu courant.

$S_4$ Alimentations et puissance dissipée. La plupart des dispositifs nécessitent une ou deux alimentations ($\pm$ 5 volt, $\pm$ 12 volt). La puissance dissipée, devient un critère déterminant si le système doit fonctionner sur alimentation autonôme (µP C.MOS). [BRI 78].

$S_5$ La compatibilité électrique. Elle est nécessaire avec les circuits logiques environnants, dispositifs C.MOS compatibles TTL en particulier.


Sélection
phototype
petite série
grande série
facteur de répétitivité
phototype
petite série
grande série
4 bits
8 bits
12 bits
16 bits
longueur de mot
parties
linéaires

Avec une classification aussi simple, nous pensons qu'à chaque étape, l'ensemble solution est réduit quant au nombre de circuits.

Lorsqu'on arrive au niveau le plus bas, il ne reste qu'un nombre très limité de circuits adéquats. Evidemment si l'ensemble solution s'avère vide, il faut réexaminer les critères secondaires.

Il y a beaucoup trop de documentations à consulter pour appliquer manuellement cette hiérarchisation comme méthode de choix. Même si ce travail est facilité par l'existence des tableaux résumés, et que les concepteurs tiennent compte des critères non techniques que nous avons exclus, en pratique.

Des algorithmes permettant d'évoluer rapidement dans la foule d'informations que sont les caractéristiques des µP's existent. Notamment celui de MAYMON/TABAK [MAY 78] et celui de BARTON/DAGLESS [BAR 77].

## 6.2 ALGORITHMES NUMERIQUES DE SELECTION DE MATERIEL EXISTANTS

6.2.1 L'algorithme de ces derniers permet d'étudier les instructions des µP au détail (voir ci-dessous), et donne les possibilités du µP sous forme graphique. C'est un soucis légitime puisque souvent les constructeurs avancent le nombre d'instructions comme caractéristique de leur µP, et il n'est pas évident de le considérer comme critère de choix. Il y a certaines instructions qui en cachent d'autres [WEI 78] ; et ce n'est pas parce qu'un µP ne possède pas tel type d'instruction que le traitement souhaité n'est pas réalisable. Enfin un nombre réduit d'instructions n'est ni un critère de puissance, ni un critère d'adéquation avec les besoins de l'utilisateur.

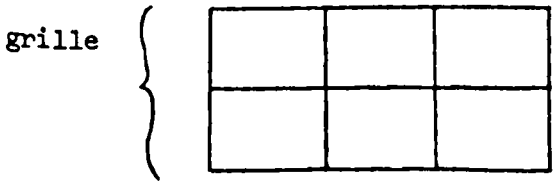
Les auteurs divisent l'ensembles des instructions d'un µP en deux groupes :

les instructions opérationnelles et
les instructions de contrôle (retours, sauts, appels de sous-programmes)

Chaque instruction de µP peut s'exprimer sous l'une des 3 formes suivantes :

Op1 <—— Op1 . u
Op1 <—— Op2 . u
Op1 <—— Op1 . b . Op2

où Op1 et Op2 sont des opérandes, u et b étant des opérateurs uninaires et binaires, respectivement. Il s'ensuit que chacune de ces formes peut être représentée par des entrée dans une grille bidimentionnelle :

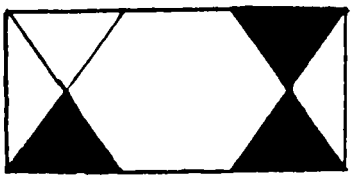



longueur de l'instruction : 1 2 ou 3 ..... octets

une grille vide signifie : instruction inexistente ; une grille avec une croix signifie instruction impossible. Dans la grille, une case vide signifie pas d'opérande, un triangle vide : pas de choix, et un triangle plein opérande avec possibilité de choix : Les triangles sont utilisés pour interprêter les instructions de contrôle.

Par exemple la grille ci-dessous signifie

. pas de choix pour l'opérande 1, mais choix possible pour Op2
. instruction sur 2 octets inexistente
. choix possibles pour Op1 et Op2 sur 3 octets.



Nous donnons ci-dessous un tableau résumant les caractéristiques des instructions opérationnelles du 8080 d'INTEL, et la légende permettant de le comprendre.

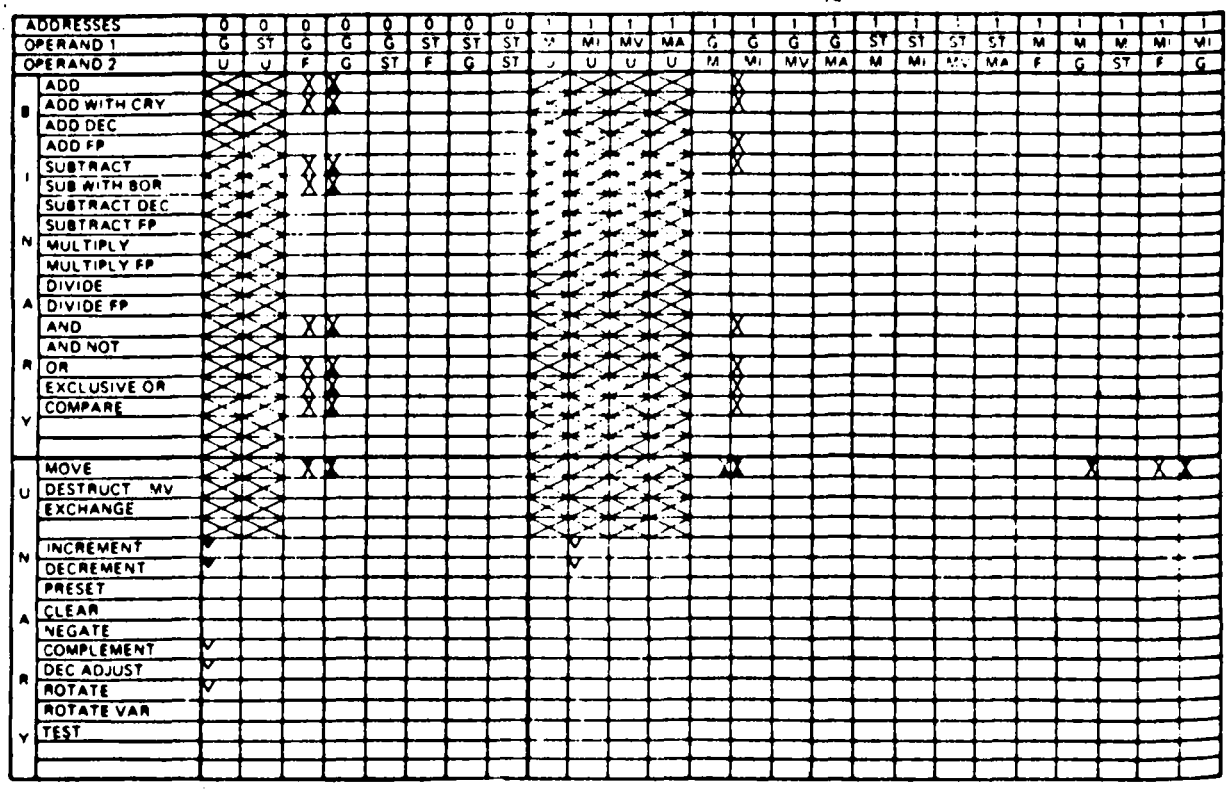
2

| Symbol | Name | Addressing mechanism: No 'choice' | Addressing mechanism: 'Choice' |
|---|---|---|---|
| F | Immediate | Implied field in instruction | Never |
| G | Register | Implied register | Register selected by F |
| M | Memory | Implied memory location | Memory location selected by F |
| MI | Indirect | Memory location selected by M or by implied G | Memory location selected by choice of G |
| MV | Indexed | Memory location indexed by implied index register | Memory location indexed by choice of index registers |
| MA | Autoincrement/autodecrement | Software-implemented stack with implied stack pointer | Software-implemented stack with choice of stack pointers |
| ST | Hardware stack | Implied hardware stack | Choice of hardware stacks |

U denotes unary operations, i.e. OP2 is null

## Légende des Opérandes

**Intel 8080** Tableau des instruction opérationnelles.



Nous ne nous étendrons pas davantage sur l'algorithme de BARTON, un tableau comparatif nous permettra de le comparer à celui de MAYMON, résumé ci-après.

### 6.2.2 L'algorithme de MAYMON-TABAK

Cet algorithme a pour but d'aider le concepteur à choisir un équipement, par exemple le microprocesseur. Naturellement, on suppose disposer : d'une part des caractéristiques de certains microprocesseurs, d'autre part de contraintes déduites de l'application.

L'algorithme se divise en deux parties :

#### 6.2.2.1 L'approximation des caractéristiques par des courbes

Il y a autant de courbes $f_j$ que de caractéristiques retenues pour l'étude. Par exemple on a les courbes représentant :

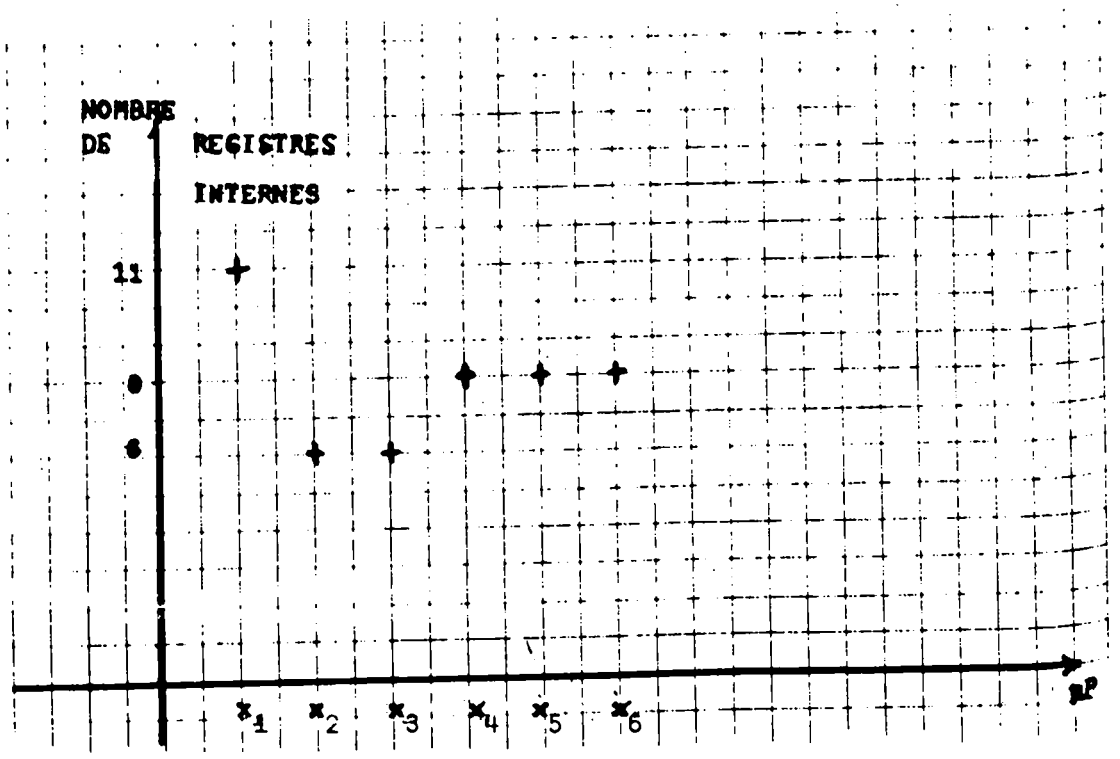
- le prix,
- la durée moyenne d'une instruction en µseconde,
- le nombre d'instruction,
- le nombre de registres internes, etc...

A chaque µP on associe une abscisse $x_i$ ; par exemple
pour l'INTEL 8080, on aura i = 1,
pour le M 6800 , on aura i = 2,
pour le F8 , on aura i = 3.

i entier positif.

L'attribution d'un indice à un µP n'est pas arbitraire ; par exemple pour les prix, elle sera fonction des performances du µP.

Ainsi, on a graphiquement, pour le nombre de registres internes des 6 µP's étudiée dans [MAY 77] :




Sur l'axe des ordonnées on porte les valeurs quantifiées de la propriété étudiée, par µP. Enfin on cherche par des "méthodes de régression à obtenir la courbe ($f_4$ ici) autour de laquelle s'articulent ces points. Les résultats obtenus et les données de départ sont à la page suivante.

6.2.2.2 L'optimisation des fonctions

Disposant de cet ensemble F de fonction : $\{f_1, f_2, f_3, \ldots, f_4\}$, on peut alors y appliquer les choix dictés par l'application.

On peut ainsi chercher à minimiser le temps moyen d'une instruction, celui des sources de tension ou même le coût. On écrit alors la combinaison linéaire :

$$\min_{x \in R} J(x) = \omega_1 f_1 + \omega_2 f_2 + \omega_3 f_3 + \omega_4 f_4 + \omega_5 f_5 + \ldots.$$

Le critère d'adéquation ainsi formé est une somme pondérée des fonctions de F. Le poids de chaque coefficient étant établit à partir de l'importance relative accordée à la propriété en question.

| Property / System | (1) Cost, $ (Spring 1976 prices) CPU + Timing Unit | (2) Mean Instruction time, μsec | (3) No. of Instructions | (4) No. of Registers | (5) No. of Accumulators | (6) No. of Voltage Sources | (7) No. of Languages |
|---|---|---|---|---|---|---|---|
| 1. 8080 | 90.- | 4.5 | 111 | 11 | 1 | 3 | 4 |
| 2. M6800 | 55.- | 5.0 | 72 | 6 | 2 | 1 | 3 |
| 3. P-8 | 65.- | 6.0 | 70 | 6 | 1 | 2 | 2 |
| 4. CP1600 | 95.- | 2.4 | 87 | 8 | 1 | 3 | 2 |
| 5. PACE | 105.- | 9.0 | 45 | 8 | 4 | 2 | 5 |
| 6. LSI-11 | 990.- | 15.0 | 400 | 8 | 1 | 2 | 2 |

Table 1. : Properties of the systems under consideration.

Cost function:

$$f_1(x) = -1010. + 2400.08x - 1948.96x^2 + 726.67x^3 - 126.04x^4 + 8.25x^5$$

Instruction time function:

$$f_2(x) = 75. - 154.95x + 119.68x^2 - 41.37x^3 + 6.52x^4 - 0.38x^5$$

No. of instructions function:

$$f_3(x) = -449. + 1353.80x - 1164.75x^2 + 442.75x^3 - 76.75x^4 - 4.95x^5$$

No. of registers function:

$$f_4(x) = 16. + 0.07x - 9.08x^2 + 4.87x^3 - 0.92x^4 + 0.06x^5$$

No. of accumulators function:

$$f_5(x) = 4. - 11.25x + 13.29x^2 - 6.17x^3 + 1.21x^4 - 0.08x^5$$

No. of voltage sources function:

$$f_6(x) = 8. - 4.95x - 1.54x^2 + 1.92x^3 - 0.46x^4 + 0.03x^5$$

No. of programing languages function:

$$f_7(x) = 17. - 28.65x + 22.96x^2 - 8.75x^3 + 1.54x^4 - 0.10x^5$$

Par exemple si un coût minimal est plus important que les autres critères, on donne à la fonction coût un poids supérieur à celui des autres fonctions.

Ensuite on délimite la région R où la contrainte doit être définie. Deux aspects peuvent intervenir : la caractéristique elle-même, et les contraintes du cahier des charges.

Par exemple, sur le graphe ci-dessus, on voit que pour $f_4$ on a : $5 < f_4 < 12$. C'est une limite indépendante de l'application, qui peut exiger que le nombre de registres soit supérieur à 8 : $f_4 \geq 8$.

Dans la référence citée, les auteurs indiquent :

$0 \leq f_1(x) \leq 150$ $,
$0 \leq f_2(x) \leq 10$ µsec.,
$30 \leq f_3$ nombre d'instructions
$5 \leq f_4$ nombre de registres
$1 \leq f_6 \leq 3$ nombre d'alimentations
$1 \leq f_7$ nombre de langages

Ce qui leur permet d'obtenir $R = \{x \mid 1 \leq x \leq 3 \text{ et } 4 \leq x \leq 5\}$
et d'avoir pour fonction à minimiser

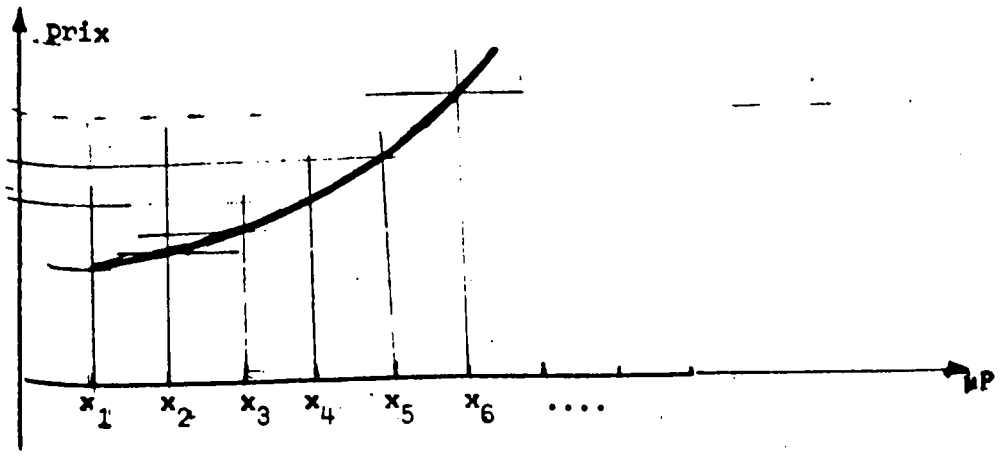
$$J(x) = \omega_1 f_1 + \omega_2 f_2 + \omega_6 f_6.$$
$$x \in R$$

avec $0 \leq \omega_i \leq 1$ $\forall i$ et $\sum_i \omega_i = 1$.

6.2.2.3 Critiques sur l'algorithme de Tabak-Maymon

L'aspect le plus discutable dans cet algorithme est la justification que donnent les auteurs sur la numérotation des microprocesseurs.

On conçoit très bien que les prix puissent suivre une courbe d'allure ci-dessous, les $x_i$ les plus faibles correspondants aux µP les moins sophistiqués.




Il est difficile d'appliquer objectivement la même justification aux autres critères. D'ailleurs les auteurs avouent se baser sur leur expérience [TAB 72]. Donc les fonctions obtenues n'ont rien d'intrinsèque car elles dépendent de la numérotation des µP's.

On pourrai penser à faire une numérotation arbitraire, car par les 6 points de l'exemple, on peut toujours faire passer un polynôme de degrè cinq. Mais est-ce là la meilleure représentation du critère ; qui par ailleurs n'aura de signification qu'aux valeurs entières de x ?

Enfin, même si les valeurs données aux coefficients $\omega_i$ permettent de donner plus (ou moins) d'importance à un critère, une nette hiérarchisation entre les critères n'apparait pas. Les auteurs n'indiquent pas pourquoi ils prennent en compte le nombre d'instructions d'un µP, plutôt que la longueur de mot, par exemple.

## 6.3 PROPOSITION D'ALGORITHME SIMPLIFIE DE CHOIX DE MICROPROCESSEUR

Bien sûr, l'objectif de cet algorithme simplifié est de limiter la portée des critiques faites à celui de MAYMON.

a) Les critères seront examinés dans l'ordre décroissants de leur importance.

b) La numérotation des microprocesseurs est arbitraire et nous ne cherchons pas à représenter graphiquement une propriété. Dès lors il n'y a plus de problème d'approximation d'une série de points par une fonction monotone.

Etapes à suivre :

1 - Examen des critères de premier niveau.
2 - Examen des critères de deuxième niveau.
3 - Mise sous forme de tableau des autres critères pour l'application seulement (ceux qui sont quantifiables).

Les données de la tables sont des grandeurs normées.

4 - $\begin{cases} \text{la fonction à minimiser est} \\ J_i = \sum_{j=1}^{n} \omega_j f_{ij} \quad \text{avec} \quad \sum_{j=1}^{n} \omega_j = 1 \\ i \in \{1,\ldots,m\} \end{cases}$

par exemple $J_1(x) = \omega_1 f_{11} + \omega_2 f_{12} + \omega_3 f_{13} + \omega_4 f_{14} + \omega_5 f_{15}$

R et $\omega_j$ jouant le même rôle que dans [MAY 78]

avec $0 \leq \omega_j \leq 1$.

| Property / System | (1) Cost, $ (Spring 1976 prices) CPU + Timing Unit | (2) Mean Instruction time, μsec | (3) No. of Instructions | (4) No. of Registers | (5) No. of Accumulators | (6) No. of Voltage Sources | (7) No. of Languages |
|---|---|---|---|---|---|---|---|
| 1. 8080 | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ | $f_{15}$ | $f_{16}$ | $f_{17}$ |
| 2. M6800 | $f_{21}$ | $f_{22}$ | $f_{23}$ | $f_{24}$ | $f_{25}$ | $f_{26}$ | $f_{27}$ |
| 3. F-8 | $f_{31}$ | $f_{32}$ | $f_{33}$ | $f_{34}$ | $f_{35}$ | $f_{36}$ | $f_{37}$ |
| 4. CP1600 | $f_{41}$ | $f_{42}$ | $f_{43}$ | $f_{44}$ | $f_{45}$ | $f_{46}$ | $f_{47}$ |
| 5. PACE | $f_{51}$ | $f_{52}$ | $f_{53}$ | $f_{54}$ | $f_{55}$ | $f_{56}$ | $f_{57}$ |
| 6. LSI-11 | $f_{61}$ | $f_{62}$ | $f_{63}$ | $f_{64}$ | $f_{65}$ | $f_{66}$ | $f_{67}$ |

m est le nombre d'élément parmi lesquels il faut choisir
n est le nombre de propriété retenues.

Exemple :

Supposons, en utilisant les éléments du tableau précédent, que m est égal à 3, nombre de µP's parmi lesquels nous pouvons choisir.

Soit n = 7 le nombre de propriétés retenues.

Nous voulons déterminer la fonction J qui ait le coût minimum. Comme nous travaillons dans un ensemble fini, nous allons minimiser J telle que :

$$J = \begin{Bmatrix} 50 & 4.5 & \ldots & 3 & 4 \\ 55 & 5 & \ldots & 1 & 3 \\ 65 & 6 & \ldots & 2 & 2 \end{Bmatrix} \begin{Bmatrix} \omega_1 \\ \omega_2 \\ \cdot \\ \cdot \\ \cdot \\ \omega_7 \end{Bmatrix}$$

| Property / System | (1) Cost, $ (Spring 1976 prices) CPU + Timing Unit | (2) Mean Instruction time, µsec | (3) No. of Instructions | (4) No. of Registers | (5) No. of Accumulators | (6) No. of Voltage Sources | (7) No. of Languages |
|---|---|---|---|---|---|---|---|
| m1. 8080 | 50 | 4.5 | 111 | 11 | 1 | 3 | 4 |
| m2. M6800 | 55 | 5.0 | 72 | 6 | 2 | 1 | 3 |
| m3. F-8 | 65 | 6.0 | 70 | 6 | 1 | 2 | 2 |

La pondération, c'est-à-dire le choix des $\omega_j$, a une influence très nette sur le choix final, comme on peut le voir sur les choix obtenus, en 3 essais, avec des $\omega_j$ différents.

| | $\omega_1$ | $\omega_2$ | $\omega_3$ | $\omega_4$ | $\omega_5$ | $\omega_6$ | $\omega_7$ | CHOIX |
|---|---|---|---|---|---|---|---|---|
| 1er essai | 0,4 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 6800 |
| 2e essai | 0.1 | 0,1 | 0,4 | 0,1 | 0,1 | 0,1 | 0,1 | 8080 |
| 3e essai | 0,1 | 0,1 | 0,1 | 0,4 | 0,1 | 0,1 | 0,1 | 8080 |

$$\text{Avec } J = \begin{Bmatrix} J_1 \\ J_2 \\ J_3 \end{Bmatrix} \quad \begin{Bmatrix} 43,35 \\ 30,9 \\ 34,1 \end{Bmatrix} \quad \begin{Bmatrix} 51,75 \\ 36 \\ 36,2 \end{Bmatrix} \quad \begin{Bmatrix} 21,75 \\ 16,2 \\ 17 \end{Bmatrix}$$

$1^{er}$ essai $\quad 2^{e}$ essai $\quad 3^{e}$ essai

Au troisième essai par exemple, le choix est le 8080 d'INTEL car la valeur $J_1 = 21,75$ la plus grande, correspond au µP ayant le plus grand nombre de registres internes (poids $\omega_4 = 0,4$).

Après avoir brièvement présenté deux algorithmes de sélection de microprocesseurs, nous avons fait une proposition d'algorithme de choix simple. Le tableau ci-dessus en est un résumé et permet des comparaisons.

En fait à partir d'un certain nombre de fonctions d'évaluation (ou propriété) des microprocesseurs, nous essayons d'en optimiser la combinaison linéaire J.

Chaque µP est supposé être décrit dans la base de données de COSMIC, à l'aide de ses caractéristiques.

Cela n'a aucune importance que ces caractéristiques se représentent graphiquement. Le problème est celui du compromis entre un certain nombre de critères qui peuvent se mesurer par des fonctions.

Mais tout n'est pas de choisir le microprocesseur, il faut lui adjoindre d'autres boîtiers (cf. Chap. 4). Vu la diversité qui existe dans le domaine ; et le fait que certains µP's ont dans leur famille une gamme suffisante de circuits d'interface compatibles, nous n'avons examiné que le choix des mémoires.

| | BARTON-DAGLESS | HAYMON-TABAK | NOTRE PROPOSITION |
|---|---|---|---|
| *Critères de comparaisons examinés* | *Ensemble des instructions.* On voit mal comment seront analysés les autres critères. | Tous sans hiérarchie | Tous hiérarchisés |
| Mise en oeuvre | . très lourde<br>. un µP analysé à la fois<br>. beaucoup trop de données à la fois<br>. non conversationnelle | . à tous niveaux<br>. peut lourde tous les µP's sont examinés à la fois, toutes les caractéristiques aussi<br>. conversationnelle | . sur critères secondaires pour l'application seulement.<br>. facile<br>. conversationnelle<br>. à chaque étape ou cerne mieux la solution |
| Exploitation des résultats | . pénible<br>. sous forme de tableaux saturés | simple | simple et synthétique |
| Utilisation plutôt pour | comparaison des microprocesseurs | . comparaison<br>. adéquation µP-application | adéquation µP-application |
| Justification | objective basée sur l'examen objectif des ≠ instructions | basée sur l'expérience donc subjective | importance nette de la pondération sur la fonction à minimiser |
| Autres remarques | il est impossible d'examiner tous les µP d'où la lenteur du choix | | on aboutit plus rapidement à l'ensemble solution |

## 6.4 LE CHOIX DES MEMOIRES

6.4.1 Ici, le critère le plus important est celui de la permanence des informations. D'aucun l'aborde sous l'aspect faible consommation de courant système.

En général, quand on travaille sur secteur, on ne recherche pas la faible consommation, mais plutôt à se prémunir contre les défauts du secteur (microcoupures, coupures, etc.).

Tandis que lorsqu'on est "embarqué", on aura travailler avec une batterie. L'alimentation par batteire des mémoires statiques CMOS est facile, tandis que pour les RAM dynamiques, elle est possible mais for couteuse.* Il se trouve que souvent on élimine les RAM dynamiques quand il y a des problèmes de permanence d'informations, car la batterie doit, avec ces mémoires, continuer à rafraîchir.

Les RAM statiques ne nécessitent pas de rafraîchissement et une seule source d'alimentation est suffisante. Elles peuvent être maintenues sur batterie et ne coûte pas aussi cher que les RAM dynamiques.

6.4.2 Le deuxième critère, déductible du cahier des charges, est la capacité de mémoire vive exigée par le µP d'une part ; et celle nécessaire pour stocker en mémoire morte les quantités d'informations mises en jeu d'autre part.

a) Lorsque la capacité est supérieure à 10 KØ, l'utilisation des RAM dynamiques devient une solution moins onéreuse.

En effet les circuits de rafraîchissement sont amortis sur cet important plan de mémoire. Pratiquement, il est conseillé : [LIL 77]

- des RAMS statiques directement organisées en mots pour moins de 1 KØ
- des RAMS statiques à sorties sur un bit et montés en parallèle, entre 1 KØ et 4 KØ de mémoire.
- des RAMS dynamiques de 4 KØ.

* sauf pour le Z 80 qui est doté de circuits de rafraîchissement internes.

b) La mémoire morte (ROM) convient, pour le stockage des données, aux applications de grande série ; pendant que la PROM ou la REPROM sont destinés aux applications moyennes ou petite série.

- Le choix entre PROM et REPROM se fait en fonction des paramètres vitesse, prix et souplesse d'utilisation. Nous ne nous attardons pas sur les caractéristiques et les différences entre ROM, REPROM, PROM, EPROM car le sujet est traité dans [LIL 77].

- Chaque fois qu'on a besoin de mémoire morte en phase d'étude ou de mise au point, c'est la EPROM qui convient le mieux, car comme son nom l'indique, elle est effaçable. On peut aussi utiliser des boîtiers EPROM chaque fois qu'un équipement spécifique aura besoin de voir son programme parfois modifié.

6.4.3 La vitesse, le prix et la souplesse d'utilisation sont les 3 autres critères déductibles pour les mémoires (vives ou mortes).

a) Pour les mémoires vives, le choix peut être techniquement guidé par le tableau suivant.

| Type de RAM | faible consommation | densité | prix le plus bas | vitesse | facilité d'utilisation |
|---|---|---|---|---|---|
| STATIQUES P MOS N MOS > 4 KØ | + | + | + | + | ++ |
| STATIQUES C MOS | +++ | - | -- | - | ++ |
| STATIQUES SCHOTTKY ECL | - | - | -- | +++ | +++ |
| DYNAMIQUES < 4 KØ | ++ | ++ | - | ++ | - |
| > 4 KØ | ++ | ++ | + | ++ | - |

c'est ainsi que de grandes vitesses imposent l'utilisation des RAM bipolaires statiques.

b) Pour les mémoires de programmes : lorsque c'est la vitesse qui prime, (comme par exemple une mémoire de contrôle de µP bipolaire microprogrammable), la bonne solution réside dans le choix de boîtiers à technologie bipolaire (ROM, PROM).

S'il n'y a pas d'exigence particulière (temps d'accès supérieur à 500 ns) et lorsque le prix est un facteur déterminant, la ROM en technologie MOS, qui permet à surface égale d'avoir une intégration 4 fois plus grande, semble plus avantageuse.

La sélection des composants d'un système à base de µP serait incomplètement traitée si nous n'examinons pas ce que sont les standards ; car leurs objets est de faciliter cette conception et surtout la réalisation.

## 6.5 LES STANDARDS

. Par définition, un standard correspond à une norme dont le but est d'en améliorer l'utilisation. Ainsi l'équipement dit standard sera de mise en oeuvre rapide et aisée.

. Ici les standards sont des objets :

- les plus utilisés (: sur plusieurs µP's différents, dans une large gamme d'applications) ;

- et sont parfaitement décrit (documentation).

. Avant d'adopter un standard particulier, il faut que le concepteur sache :

a) ce qu'il y gagne (temps de développement par exemple) ; et à quel niveau (taille, efficacité, portabilité, indépendance par rapport au constructeur).

b) comment il est défini : sa structure, comment il s'integrerait dans sont SBµP ; où et par qui avoir des détails sur son fonctionnement.

• Les standards s'articulent autour du microprocesseur, (voir Chapitre 4), ce sera un bus, une interface, voire un système d'enregistrement de données.

Des efforts de normalisation ont même été fait au niveau du logiciel (ordinogrammes ou diagrammes logiques, langages) et au niveau des types de configurations de SBμP.

Dans la notion de standard, nous excluons l'idée de grande famille compatible. En effet certains μP's ont été cités dans la littérature comme étant des standards ; car non seulement ils peuvent être utilisés dans la plupart des applications de SBμP, mais aussi : leurs successeurs dans la même famille, utilisent les mêmes instructions, en introduisent de nouvelles, et donnent la possibilité de doubler la longueur de mot.

Exemple : Famille INTEL 8080, 8085, 8086.

Nous les appelons les μP's les plus "populaires" ; ce sont le 8080A, le M6800, le Z80, le F8.

Que ce soit un bus, ou une interface, le standard fait intervenir quatres concepts, lorsqu'il faut l'intégrer au SBμP :

a) mécanique (connections, cables, mode de branchement étoilé ou linéaire).

b) électrique (niveaux des signaux, courant débité).

c) fonctionnel (utilisable des fils de liaison, timing, protocoles).

d) opérationnel.

l'aspect opérationnel dépend des appareils utilisés et du système à réaliser ; il met en jeu la façon dont un instrument utilise l'interface. Ceci est étroitement lié au pro-logiciel.

### 6.5.1 Les bus standards

Beaucoup de bus parallèles ont été proposés en vue d'en faire des standards. Moyennant un minimum d'adjonctions, ils peuvents être utilisés sur des µP's différents.

Ce sont le S 100, le S 550, le bus du M 6800, Multibus ; Les plus "populaires" étant le S 100 d'Altair et le Multibus d'Intel.

D'autres bus sont en cours d'élaboration, comme celui que le Mullard Space Laboratory développé à Londres, à L'Université College.

Enfin la société Yrel fournit actuellement un nouveau bus, le STD 7000, pour les µP's 8 bits les plus "populaires".

### 6.5.2 Les interfaces standards

Certains auteurs, lorsqu'ils présentent les standards, mettent sous la même rubriques les bus et les interfaces standards, parce que ces dernières ont aussi des fonctions de bus. Nous préférons séparément les bus propremments dits et les interfaces.

Ces dernières ont rencontré plus de succès (réalisation, adoption) que les bus. Nous allons les citer et donner brièvement leurs caractéristiques. Les détails étant accessibles dans [LES 78] et chez le constructeur lui-même.

A partir de ces caractéristiques (les 4 concepts) il revient au concepteur, en s'aidant du tableau succint ci-dessous, de choisir le standard qui pourrait le mieux s'adapter à son SBµP.

Les cases vides du tableau sont celles où il y aurait trop d'informations à indiquer.

| Interfaces | Concepts mécaniques | Electriques | fonctionnels | Opérationnels |
|---|---|---|---|---|
| RS 232 C | 25 broches | Impulsions de + 12 V pour transférer les informations | Interface série transmettant des signaux comme des tensions | Communications asynchrones distance et vitesse limitées 15.3 m 20 Kb/S avec modem standard |
| RS 422 et 3 | | - immunité du bruit du canal différentiel<br>- réduction du bruit | | grandes distances et vitesses 1220 m, 10 Mb/s |
| 20 m A loop | | | série | lien avec télétype en local |
| IEEE 488 1975 ou HPIB | 15 instruments connectables | mise en oeuvre compliquée | parallèle contrôle local en instrumentation le plus utilisé. | - systèmes d'acquisitions de données intelligents. - distance 10 m ; vitesse 1 MB/S. - connexion entre eux de plusieurs systèmes ≠. - technique de poignée de main. |
| V 24 | | inutile à donner ici sans le schéma | série. 2 catégories de liaisons normalisées : - 39 circuits pour utilisations générales.<br>- 13 pour appel automatique. | Communication des données dans les réseaux publics |
| CAMAC ou IEEE 583 | 24 modules connectables maxi connexion simple spécifications rigoureuses | signaux TTl<br>4 tensions ± 6 V<br>± 24 V<br>stabilité<br>régulation<br>*suppression des états transitoires définis* | bus.parallèle cablé en fond de châssis.<br>3 lignes de contrôles<br>5 lignes commande<br>*24 pour Ecriture*<br>*24 pour Lecture*<br>*4 / mots d'états.* | Tous domaines des interfaces instrumentaux. (nucléaire par ex.). Vitesse $24.10^6$ bits/s<br>CAMAC c'est un ensemble de règles, un langage aussi. |

### 6.5.3 Les circuits dédiés

Définition : Un circuit dédiés est un circuit construit pour une application particulière des micro-processeurs. Nous distinguons dans cette catégorie :

. Les circuits existants, donc prédiffusés.

. Les circuits à la demande.

Ainsi on rencontrera des cartes destinées au télé-communications, à l'instrumentation médicale, à la mesure, au jouet, à l'horlogerie, etc.

Leurs avantages est qu'ils sont optimisés pour l'application cible :

- hautes performances
- très haute fiabilité.

Il revient au concepteur de vérifier si les interfaces proposées sont adaptées à son problème.

Nous donnons ci-dessous un tableau comparatif entre les standards et les circuits dédiés, ou sont résumés leurs avantages et les inconvénients.

| | | Avantages | Inconvénients |
|---|---|---|---|
| standards | | faible fiabilité<br>prix du circuit élevé. | Délai de réalisation faible; mise au point peu coûteuse. |
| circuits dédiés | circuits prédiffusés | performance limitée (fréquence); problèmes d'interface; grandes séries | bas prix<br>délais courts |
| | circuits à la demande | délai long<br>séries moyennes | circuit optimisé :<br>très haute fiabilité<br>hautes performances<br>haute intégration<br>interface adaptées au problème |

### 6.5.4 Où en est la standardisation au niveau du logiciel

. Des efforts justifiables semblent s'orienter surtout vers la standardisation du logiciel pour le temps réel.

Ces langages et les moniteurs associés se veulent d'apporter des solutions au niveau du respect des contraintes de temps.

La plupart des moniteurs temps réel offrent un certain nombre de services relativement à la prise en compte du temps :

- lancement cyclique des tâches
- activation après un certain délai
- modification des priorités des tâches.

(Ce qui entraîne souvent une durée d'exécution des programmes difficiles à prévoir temporellement).

Le principe sous-jacent à la multiprogrammation que permettent ces moniteurs est celui de priorité relative des tâches ; alors que les utilisateurs souhaiteraient raisonner sur le principe d'urgence d'une tâche.

Le vrai problème ne se situe pas seulement, à notre avis au niveau d'un langage de programmation, mais aussi dans un ensemble de techniques et de méthodes de développement de logiciel.

Il revient au concepteur de décider si l'utilisation de tel ou autre système temps réel est plus pratique pour lui. Sinon il construira lui-même : et le logiciel de base à partir de COSMIC (cf. Chap. 3) et ses programmes d'applications (chap. 2).

## CONCLUSION

Nous avons fait dans ce chapitre une proposition de choix de circuits pour la construction d'un SBµP. Quoique nous nous attachions à souligner les aspects avantageux au niveau de la technologie, sans nous occuper de l'importance qu'ont les prix, les délais de réalisations et autres contraintes commerciales.

Divers constructeurs commercialisent des modules plus complexes sous la forme de cartes assemblées et testées, groupées elles aussi en familles compatibles :

- carte µθ de base ;
- extension RAM et ROM ; interfaces ;
- entrées-sorties analogiques ou numériques...

Ainsi un système peut s'étendre par simple adjonction d'autres modules, grâce à cette conception modulaire.

Le concepteur ayant choisi au moins les circuits de base de son SBµP, il faut maintenant qu'il détermine si le matériel choisit permet la faisabilité de son application. C'est-à-dire si au niveau des temps de traitements, le SBµP est adéquat.

BIBLIOGRAPHIE

[RAP 78] Howard Raphael
*"Simplify low cost µP selection"*
E.D.N. p 85-90.

[MAY 78] E. Maymon and D. Tabak
*"Selection of microprocessor equipement"*
EUROMICRO 78, p 85-88.

[TAB 72] D. Tabak
*"Choice of complex systems based on past experience"*
Int. J. Systems S.C.L. 1972, Vol. 2, n° 4, p 411-420.

[BAR 77] Micheal Barton, Erik Dagless
*"Graphical approch to µP's comparison"*
Microprocessor, Vol. 1, n° 6, à partir de Août 77 série d'articles.

[BRI 78] Cl. Brie et M. Guivarch
*"Critères de choix des µP existantes"*
Le Nouvel Automatisme, Sept.-Oct. 78.

[LIL 77] H. Lilen
*"Mémoires intégrées"*
Edition Radio 1977.

[WEI 78] C. Dennis Weiss
*"Analysis of µP instruction sets"*
Microprocessor data manual 78, E.D.N.

[WHI 78] M. Whitbread
*"Standards in µC system Desing"*
Microprocessors and microsystems, Vol. 2, n° 6, Déc. 78.

[LES 78] Austin Leslie et Rodnay Zacks
*"Techniques d'Interfaces des µP's"*
SYBEX.

# CHAPITRE 7 : ETUDE DES CONTRAINTES DE TEMPS ET DE TAILLE

INTRODUCTION

7.1 OU APPARAISSENT LES CONTRAINTES DE TAILLE TEMPS

7.2 COMMENT POURRAIT-ON FAIRE UNE ESTIMATION DES TEMPS

7.2.1 La méthode la plus précise
7.2.2 Les 3 modes d'interaction µP-environement
7.2.3 Les difficultés rencontrées pour utiliser le modèle des files d'attente

7.3 EN QUELS TERMES S'EXPRIMENT LES CONTRAINTES DE TAILLE LOGICIEL

CONCLUSION

BIBLIOGRAPHIE

# INTRODUCTION

Les tâches, décrites dans la partie opérative (PØ) seront les tâches utilisateurs. Les tâches doivent se partager les ressources du SB P, par exemple :

- l'ensemble des registres internes ou un ensemble de mémoires tampons ;
- l'unité centrale
- les unités périphériques (disquettes etc...)
- etc ...

Il est nécessaire de faire des compromis quant à leurs tailles respectives. Comme éléments pouvant influencer cette décision, citons entre autres :

- la priorité : les tâches de hautes priorités doivent être plus courtes de façon à permettre aux tâches moins prioritaires, d'entrer en possession des ressources. Ceci pour éviter des temps morts pénalisants

- la durée : une tâche peut comporter un petit nombre d'instructions qui nécessitent beaucoup de temps pour l'exécution. C'est souvent le cas des transferts de données avec la périphérie, et la prise en compte de certains évènements.

Dans ce chapitre, nous examinons les contraintes de temps et de taille pour une application donnée.

Si ces deux variables sont quantifiées et parfaitement définies dans le cahier des charges, par l'utilisation, le problème du concepteur se réduit à rechercher la cohérence du système. Les tailles indiquées et surtout le temps donné sont alors des contraintes. Sinon il faut essayer d'en faire une estimation.
Dans tout les cas, le concepteur examine les trois situations suivantes :

a) L'utilisation d'un µP pour contrôler un phénomène physique. Cas que nous étudierons plus particulièrement ici.

b) L'utilisation d'un µP pour contrôler un ensemble de phénomènes physiques. Cas qui tombe aussi dans le cadre de notre étude.

c) L'utilisation d'une fédération de µP's pourcontrôler un ensemble de phénomènes physiques, chaque site étant du type a) ou b)

Dans le cas c), il faudra tenir compte des contraintes non négligeables relatives aux communications entre les différents sites.

Ensuite il lui faut rechercher :

1° Les endroits de l'application où peuvent apparaître les contraintes de temps,

2° comment il pourrait les prendre en compte,

3° en quels termes s'expriment les contraintes de taille de l'application en question.

## 7.1 OU APPARAISSENT LES CONTRAINTES DE TEMPS

Les contraintes peuvent apparaître à trois niveaux en entrées, pendant le traitements et en sorties.

### 1° Les entrées

Il faut par exemple connaitre :

- le débit des blocs de données sur les lignes de transmission
- celui des mots d'états venant des périphériques, et
- celui des données provenant des convertisseurs analogique-digital (CAD).

Il est possible de déduire du cahier des charges :

- les instants où les entrées seront disponibles ; comment le processeur le saura et de quelle manière il devra réagir

- la fréquence propre des données, et pendant combien de temps elles seront disponibles pour le µP.

2° Les traitements qu'effectue le µP :pour chaque tâche, d'après la définition donnée au chapitre 1, le concepteur doit recherche

. les conditions d'activation (des tâches ou interruptions)
. le modèle mathématique de l'algorithme, qui ici est souvent déterministe par opposition à probabiliste, en vue de l'ordonancement
. le délai de fin d'exécution
. la priorité, dont nous avons déjà parlé
. la possibilité ou non qu'à la tâche d'être interrompu lors de son exécution.

L'ensemble des tâches sera ainsi caractérisé par

- le nombre
- les contraintes de précédences temporelles.

Pour synthétiser la durée d'exécution d'une tâche, il faut souvent cumuler les étapes suivantes :

- réception des données (provenant de la périphérie)
- stockage des données en mémoire
- traitement
- transfert des résultats en mémoire
- transfert de ces résultats vers la périphérie.

3° Les sorties sont traitées de manière synmétrique aux entrées. Lorsque pour les entrées sont recensées les grandeurs utiles, on conçoit alors les capteurs capables de les acquérir ; pour les sorties ce sont des paramètre sur lesquels ont devra agir qu'il faut recenser. Les actionneurs qui agissent sur ces paramètres sont alors à concevoir.

Nous faisons une place particulière pour ce qu'on appelle "le temps de réponse" : car nous avons dit au premier chapitre que les systèmes qui nous intéressent sont ceux du type temps réel. Par définition, un système temps réel est celui dont, le temps de réponse est déterminé en fonction d'une contrainte extérieure (temps de service et/ou temps d'attente) imposée à priori.

C'est au niveau des tâches de test et de prise d'information que se pose ce problème.

Ces tâches permettent de prendre en compte un évènement (ça peut être une donnée) extérieur et de l'enmener jusqu'à l'organe de calcul, à l'initiative, soit de l'environnement, soit du P. Cet évènement demande (en général) dès qu'il apparait, l'exécution d'une action effective au plus tard après un délai : le temps de réponse.

Pour prendre en compte un tel énèvement, il faudra échantillonner le signal à une fréquence au moins égale à la pérennité de celui-ci.

Par exemple : un µP a pour tâche, la surveillance d'un certain nombre d'états de machine. Ces états sont des variables (booléennes) qui sont susceptibles de changer 50 fois par seconde. Si on veut surveiller ces changements d'états, en étant sûr de ne commettre aucun oubli, pour en dresser un historique (ce qui peut être utile dans le cas où beaucoup de ces variables changent d'état fréquemment), l'ensemble des tâches concernées (acquisition + détection + stockage) doit pouvoir s'exécuter au moins toutes les $\frac{1}{50}$s, soit 20 ms pour l'ensemble des variables.

En fait le concepteur étudie deux possibilités suivant qu'il peut ou non choisir ce temps de réponses. Sa valeur peut être imposée par des raisons technologiques.

Par exemple on peut savoir qu'à l'occurrence d'une interruption prioritaire, on dispose de 50 ms pour faire la lecture d'informations sur un certain nombre de registres et que passé ce délai, l'information aura disparue. L'utilisateur peut aussi savoir que la dynamique du phénomène à observer est rapide ou que la période d'échantillonnage est extrèmement courte. Nous pensons que ce genre de contraintes n'est pas discutable ; on les satisfait ou pas. Ce qui conditionne fortement le choix de tel ou autre équipement, technique (ADM par exemple) ou technologie. Une des caractéristiques les plus importantes des signaux de commande que le µP (ou le µθ) doit fournir au processus qu'il contrôle concerne le temps : quand doit-on montrer un signal ? Au bout de quel délai doit-on le redescendre ? Quand fait-on une mesure ? Avec quelle périodicité doit on acquérir une donnée.

Le temps utile pour le processus est déterminé par lui, on l'appelle le temps réel. Au contraire, le temps qui concerne le µP est caractéristique de sa vitesse de fonctionnement, notamment de la durée d'exécution d'une instruction élémentaire. On l'appèle le temps propre.

On se trouve ainsi en présence de deux acteurs :

le µP et l'environnement

qui inter-agissent l'un sur l'autre ; que ce soit dans le type d'application a) ou dans le type b).

## 7.2 COMMENT POURRAIT-ON FAIRE UNE ESTIMATION DES TEMPS ?

Tout le problème consiste à synchroniser le temps propre et le temps réel. Comme de toutes les façons un µP ne peut fonctionner autrement que selon son temps propre, il faut qu'il fournisse des commandes adaptées au déroulement du processus à contrôler. Donc des résultats justes donnés avant l'écoulement d'un délai fixé. C'est le fonctionnement en temps réel.

Cette situation est élégamment résumé dans [WIR 77], où le concepteur doit s'arranger pour que, si T(I) dénote le temps (maximum) pour exécuter l'instruction I, on ait, dans le modèle

<<producteur-consommateur>>

T (chercher ; vide ; envoyer (...) ; consommer) < T (produire)

Les deux acteurs peuvent indifféremment jouer le rôle de consommateur ou de producteur de la donnée.

Les trois règles suivantes sont celles qui permettent de calculer la durée d'une tâche simple, c'est-à-dire où n'intervient aucun périphérique.

- α Le temps d'exécution des instructions séquentielles est la somme de leurs durées.
- β Le cas le plus défavorable d'exécution d'une instruction conditionnelle est la durée de l'alternative la plus longue a exécuter.
- γ Pour les boucles, le cas le plus défavorable est le produit du nombre maximum d'itérations par la durée du corps de la boucle.

Si nous divisons l'ensemble des tâches en 3 grands groupes [GRO 79] les trois règles ci-dessus donnent les résultats d'estimation qui constituent une première approche.
Les trois groupes de tâches sont :

1° **Les tâches de commande**

Une fonction de commande représente pour l'unité centrale un travail de gestion des données, d'évaluation des états logiques et de génération d'ordres. Un grand nombre de données devront transiter rapidement par le bus.

2° **Les tâches de calculs**

La fonction calcul consiste pour le µP à exécuter un certain nombre d'opérations arithmétiques afin de déterminer les valeurs des variables. Pour le temps réel, il y a une exigence de puissance de la CPU, et une occupation des mémoires plus importante.

## 3° Les interruptions

Les tâches de test et prise d'information.

Exemple : le calcul de temps d'une tache de calcul. Langage utilisé :
Assembleur 8080 A d'Intel

| Programme | | | Explication | Cycles | Longueur |
|---|---|---|---|---|---|
| Début | LXI | H, ORG 1 | Chargement en immédiat du registre HL (origine de la zone de départ) | 10 | 3 |
| | LXI | D, ORG 2 | Chargement en immédiat du registre DE (origine de la zone d'arrivée | 10 | 3 |
| Boucle | MOV | A, M | Chargement de l'accumulateur avec le contenu de la position de mémoire, définie par HL | 7 | 1 |
| | STAX | D | Stockage du contenu de l'accumulateur dans la position de mémoire définie par DE | 7 | 1 |
| | INX | H | Incrémentation de HL | 5 | 1 |
| | INX | D | Incrémentation de DE | 5 | 1 |
| | MOV | A, L | Transfert du contenu de L dans A | 5 | 1 |
| | CPI | L FIN | Comparaison en immédiat (test de FFh pour fin de zone) | 7 | 2 |
| | JNZ | Boucle | Saut à l'adresse « boucle » | 10 | 3 |
| | MOV | A, H | Transfert de H dans A | 5 | 1 |
| | CPI | H FIN | comparaison en immédiat (test de H pour fin de zone) | 7 | 2 |
| | JNZ | Boucle | Saut à l'adresse « Boucle » | 10 | 3 |

(m fois)

Ce programme recopie une zone mémoire dans une autre zone, ces zones ayant des emplacements quelconques.
La longueur de la zone à recopier est de 256 octets.
On voit ici que le nombre de cycles est $20 + 46\,m + 22 + \frac{2m}{256}$ .

Si m = 300 et la durée d'un cycle horloge, on a comme temps d'exécution 6,9 ms pour le 8080 A avec 0,5 µC temps de cycle 8080 A. [GIR 79].

Remarque : le temps moyen d'exécution d'une instruction avec le 8080 A est de 6,4 cycles d'horloge. (minimum : 2 µs ; maximum 9 µs).

Avec le Z80 et le 8086 on a les résultats suivants, pour le même traitement :

| Z80 | | | 8086 | | |
|---|---|---|---|---|---|
| 11 00 02 | LD DE, 0200 | ; Load destination into DE | B9 FF 00 | MOV C, 00FF | ; Load byte count into C |
| 21 00 03 | LD HL, 0300 | ; Load source into HL | BE 00 03 | MOV SI, 0300 | ; Load source into SI |
| 01 FF 00 | LD BC, 00FF | ; Load byte count into BC | BF 00 02 | MOV DI, 0200 | ; Load destination into DI |
| ED B0 | LDIR | ; Block move | F2 | REP | ; Repeat primitive |
| | END | | A4 | MOVB | ; Move instruction |
| | | | | END | |
| Instructions: | 4 | | | 5 | |
| Bytes: | 11 | | | 11 | |
| Execution time (at 4 megahertz): | 1,347 microseconds | | | 1,313 microseconds | |

Nous dirons que ce genre d'estimation est "statistique".

Cette méthode est plus simple que celles des estimateurs de [RUSSEL 69] (basée sur la théorie des graphes), et [BEIZER 70] (analysant à partir de la même théorie des programme FORTRAN), [CHO 77] (il utilise un compilateur d'une extension de PLM).

Les avantages et les inconvénients de ces trois méthodes sont présentés dans [CHO 77].

*3-a) Les interruptions*

Les règles $\alpha$, $\beta$, $\gamma$ servent aussi pour calculer la durée d'une routine d'interruption . En effet celle-ci à généralement la structure suivante :

*DUREE DES DIFFERENTES PHASES DU TRAITEMENT D'UNE INTERRUPTION*

*(les microprocesseurs sont supposés fonctionner à la fréquence maximale)*

*[GIR 79]*

prise en compte et sauvegarde du contexte

| PHASE | 8080 A | | 8080 A-2 | | 8080 A-1 | | 8085 | | 8085 A | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *cycles | µs | *cycles | µs | *cycles | µs | cycles | µs | cycles | µs |
| Achèvement de l'instruction | | | | | | | | | | |
| • temps d'exécution minimum | 4 | 2 | 4 | 1,5 | 4 | 1,3 | 2 | 2 | 2 | 1,33 |
| Temps d'exécution maximum | 18 | 9 | 18 | 6,75 | 18 | 5,85 | 12 | 12 | 12 | 8 |
| Prise en compte | 11 | 5,5 | 11 | 4,125 | 11 | 3,575 | 12 | 12 | 12 | 8 |
| Sauvegarde B, C (8080) | 11 | 5,5 | 11 | 4,125 | 11 | 3,575 | | | | |
| Sauvegarde D, E (8080) | 11 | 5,5 | 11 | 4,125 | 11 | 3,575 | | | | |
| Sauvegarde H, L (8080) | 11 | 5,5 | 11 | 4,125 | 11 | 3,575 | | | | |
| Sauvegarde P S W (8080) | 11 | 5,5 | 11 | 4,125 | 11 | 3,575 | | | | |
| Temps de prise en compte mini | 11 | 5,5 | 11 | 4,125 | 11 | 3,575 | 12 | 12 | 12 | 8 |
| Temps de prise en compte maxi | 55 | 27,5 | 55 | 20,625 | 55 | 17,875 | 12 | 12 | 12 | 8 |

PROGRAMME D'INTERRUPTION

restauration du contexte

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Retour au programme principal | 10 | 5 | 10 | 3,750 | 10 | 3,250 | 10 | 10 | 10 | 6,67 |
| Restitution B,C (8080) | 10 | 5 | 10 | 3,750 | 10 | 3,250 | | | | |
| Restitution D,E (8080) | 10 | 5 | 10 | 3,750 | 10 | 3,250 | | | | |
| Restitution H, L (8080) | 10 | 5 | 10 | 3,750 | 10 | 3,250 | | | | |
| Restitution P S W (8080) | 10 | 5 | 10 | 3,750 | 10 | 3,250 | | | | |
| Temps de retour mini | 10 | 5 | 10 | 3,750 | 10 | 3,250 | 10 | 10 | 10 | 6,67 |
| Temps de retour maxi | 50 | 25 | 50 | 18,75 | 50 | 16,25 | 10 | 10 | 10 | 6,67 |

*cycles d'horloge

où la seule inconnue est le corps de la routine : les sauvegardes et restaurations étant statistiquement estimables de point de vue durée et taille.

Beaucoup de choses peuvent intervenir dans le corps d'une routine d'interruption ; et il faut aussi estimer le temps nécessaire à sa prise en compte. C'est ce que nous allons analyser au niveau des tâches de tests et prise d'informations liées aux périphériques.

*3-b)*

La fonction à réaliser peut être celle de surveillance, qui se traduit par des scrutations cycliques de variables (température, pression, tension pour des vairables analogiques : niveaux, position de vannes pour des variables logique). Le problème est de savoir combien de points on peut scruter durant une période donnée.

A chaque tâche décrivant un périphérique ou un équipement est associée une priorité. Le microprocesseur teste cette priorité pour savoir s'il doit traiter la tâche immédiatement, ou alors l'ignorer. C'est comme si chaque tâche périphérique correspondait a une routine d'interruption ; elle sera alors le cas échéant traitée de manière non interruptible, égard à sa priorité.

Chaque tâche associée à un équipement sera cyclique. Son cycle interne sera uniquement constitué par l'attente de fin de travail de l'équipement ; nous l'appelerons V(n), pour la tâche n.

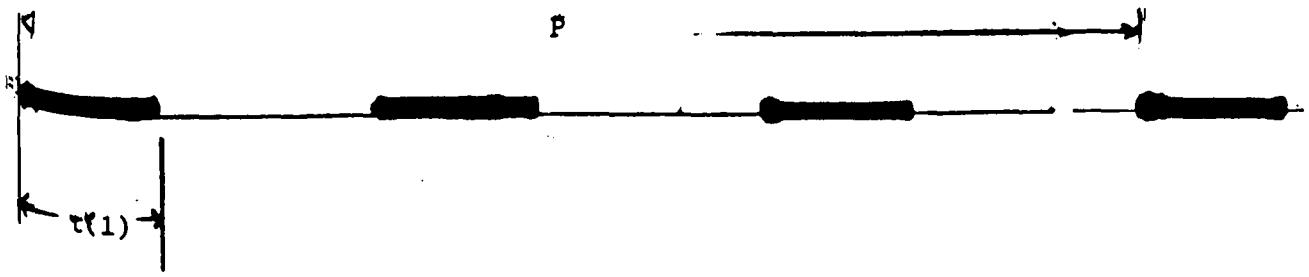
Supposons que nous ayons n niveaux de priorités pour n tâches. Les autres paramètres temporels liés au niveau de priorité n sont :

P(n) intervalle minimum séparant les demandes d'activations de niveau n, d'une même tâche

t(n) durée d'exécution de la tâche n, plus le temps nécessaire pour passer du niveau antérieur au niveau n ; et retour vers un autre niveau

Q(P,n) durée nécessaire pendant laquelle le P exécute les tâches 1 à n dans l'intervalle P

k(n) nombre maximum d'activations des tâches de niveaux n dans l'intervalle P




Exemple : avec n = 1

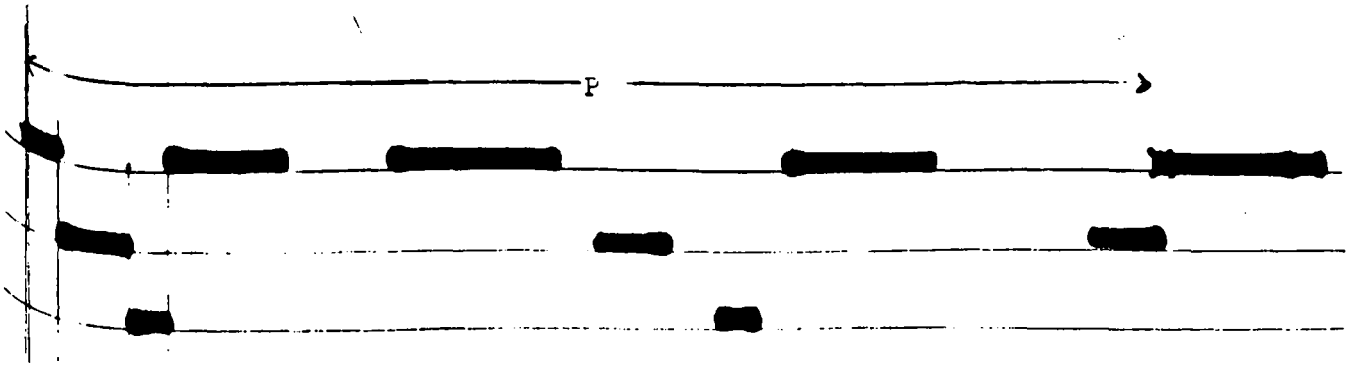
P = 300 ms

t(1) = 20 ms

Q(P,1) = t(1) * k(1) = 60 ms

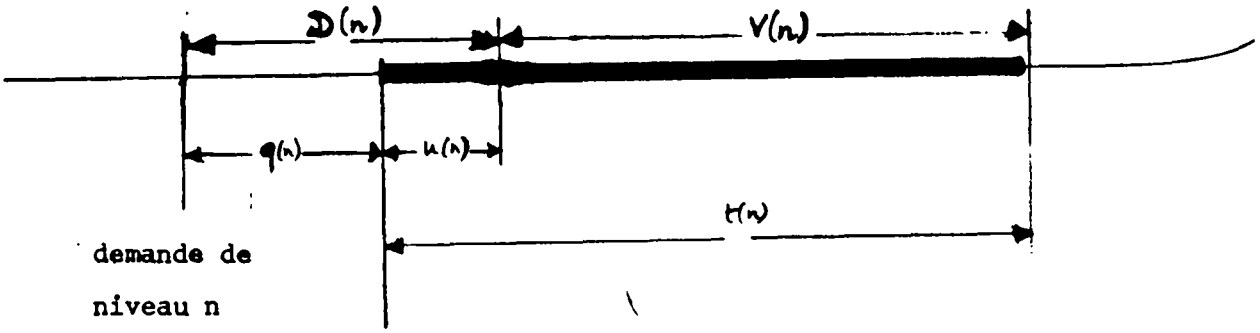
Le temps entre la première exécution et la demande d'exécution suivante peut être déterminé : si par exemple c'est un téléimprimeur que nous sommes en train de contrôler, ce délai pourrait être la durée du cycle interne de la tâche téléimprimeur.

Exemple : avec n = 3




Sur ce dernier exemple, on voit que la durée maximale de t(3) est la somme des temps d'exécution de toutes les autres tâches de priorités supérieures.

Les paramètres lié au niveau d'interruption étant déterminés, explicitons ceux qui sont liés à la prise en compte et au traitement d'une interruption :




D(n) durée totale entre la demande et la prise en compte

V(n) cycle interne du périphérique

q(n) temps passé depuis la demande pour terminer le service des tâches de niveau supérieur. (Dans l'exemple page 12, $q(n) \in [1,3,8]_{\mu s}$

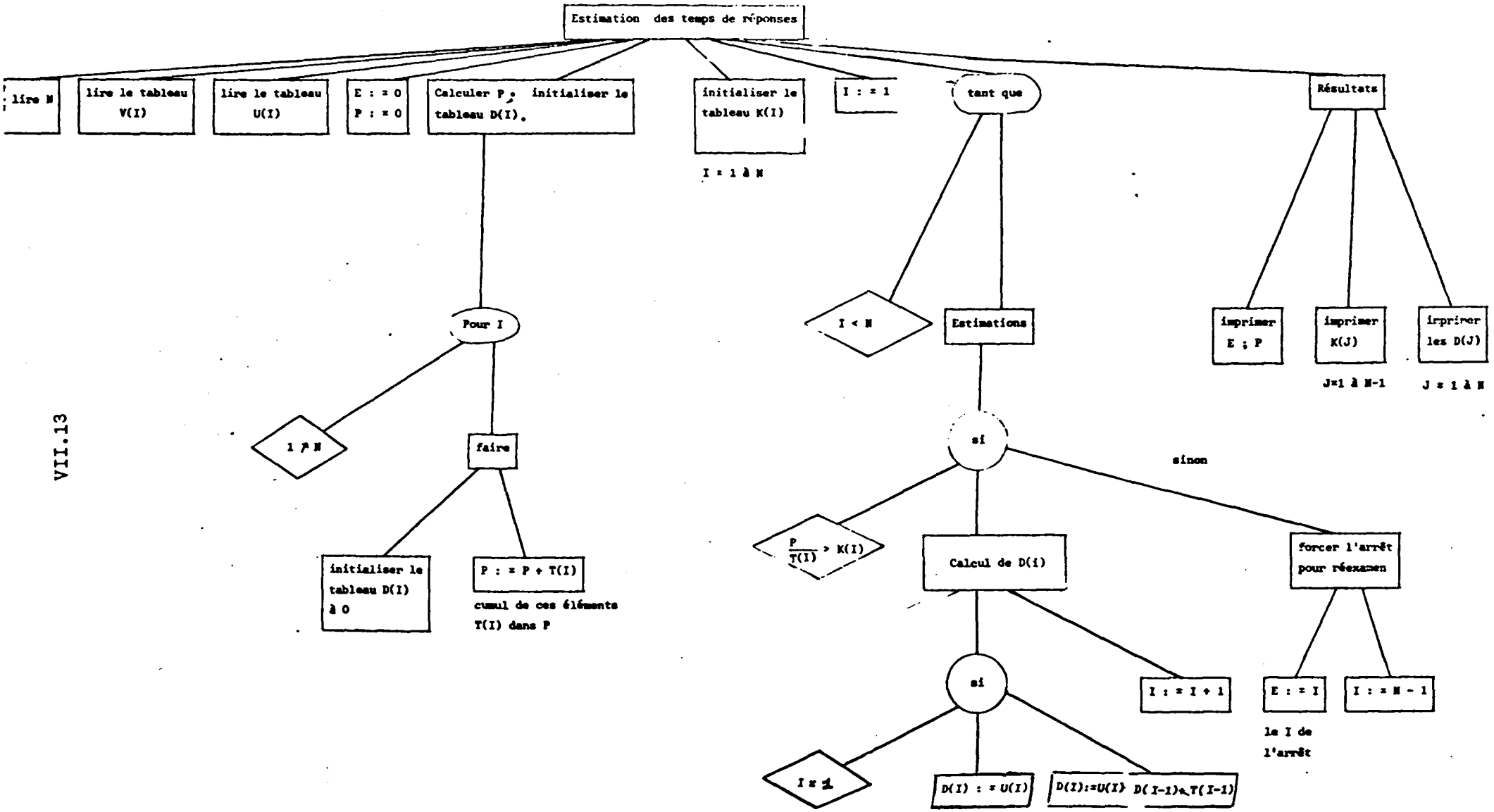
u(n) durée du changement de contexte ( et initialisation ou branchement au programme n). (u(n) = 8 µs pour le 6080 A)

Pour la tâche de niveau n, le temps disponible dans P est :

P - Q (P,n-1) ; la fraction de temps ainsi utilisable

est $\frac{Q(P,n-1)}{P} = R(n)$. Quelquefois, il faut ajuster le cycle interne du périphérique contrôlé qui n'est pas vraiment V(n), mais V(n) * (1- R(n)) ;

Si le concepteur prend un intervalle P assez grand pour qu'il contiennent plusieurs occurrences des tâches de niveau (de priorité) les moins fréquemment exécutées en temps réel (sauf l'arrêt par opérateur ou même des tâches qui ne sont exécutées qu'une fois dans la journée), on peut estimer les temps de réponses d'après l'algorithme suivant, inspiré de Cunnington. Le temps de réponse est donné par D(i) = Q(D(i), i-1) + u(i) où Q(D(i), i-1) est le temps pendant lequel le P est occupé à traiter des tâches plus prioritaires (et dont le niveau varie de (i-1) à 1) que i. i = 1,2,...,n.


Estimation des temps de réponses
lire N
lire le tableau V(I)
lire le tableau U(I)
E : = 0
P : = 0
Calculer P, initialiser le tableau D(I).
initialiser le tableau K(I)
I = 1 à N
I : = 1
tant que
Résultats
imprimer E ; P
imprimer K(J)
J=1 à N-1
imprimer les D(J)
J = 1 à N
I < N
Estimations
Pour I
1 à N
faire
initialiser le tableau D(I) à 0
P : = P + T(I)
cumul de ces éléments T(I) dans P
si
sinon
P / T(I) > K(I)
Calcul de D(i)
forcer l'arrêt pour réexamen
si
I : = I + 1
E : = I
I : = N - 1
le I de l'arrêt
I = 1
D(I) : = U(I)
D(I):=U(I) D(I-1)+T(I-1)

# EXPLICATION DE L'ARBRE DE LA PAGE PRÉCÉDENTE.

| | |
|---|---|
| i | correspond à la priorité d'une tâche |
| n | niveaux de priorités en tout |
| T | est un tableau qui contient les sommes u(i) + V(i) = t(i) On trie ce tableau pour classer les durées des tâches par ordre croissant |
| K(i) | est le nombre de fois que la durée T(i) intervient dans Q |
| P | est défini dans le programme |

- initialisation de P

$$P = T(1) + T(2) + \ldots + T(n\ ) ;$$

- initialisation du tableau K(i) les n éléments sont initialisés à 1 au moins on s'assure ainsi que chaque tâche s'exécutera au moins une fois.

Il est possible qu'une interruption de niveaux supérieur à n apparaisse plus d'une fois pendant Q(D(n), n-1) ; dans ce cas il faut lui ajouter la durée de cette tâche interruption (pas 2 de l'algorithme page précédente).

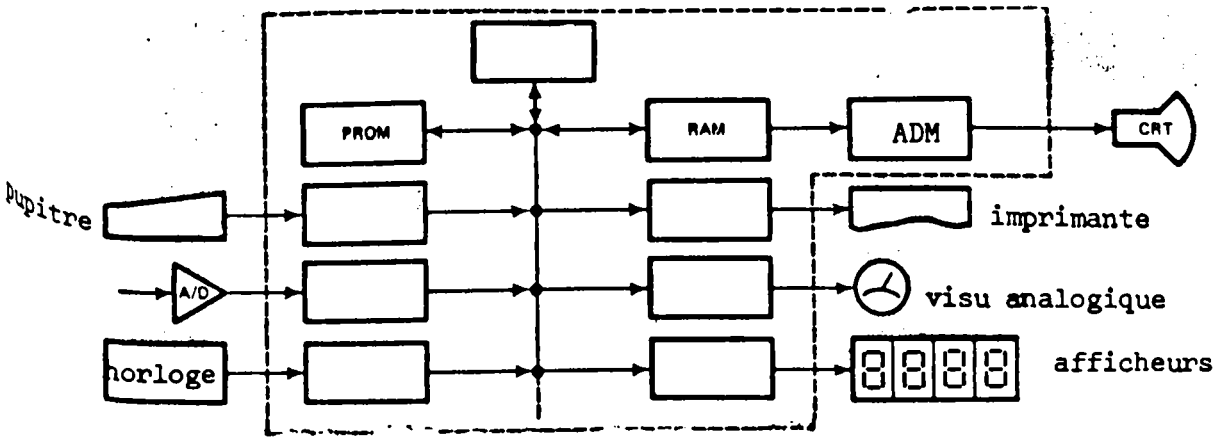
Si le temps de réponse maximum toléré pour chaque niveau est trop grand, alors on peut re-distribuer les niveaux d'interruptions entre tâches, ou réécrire les programmes pour qu'ils durent moins longtemps, ou les deux. En pratique, il y a plusieurs périphériques, chacun travaillant à sa propre vitesse ; et certains demandant un temps de réponse très rapide.

Exemple.

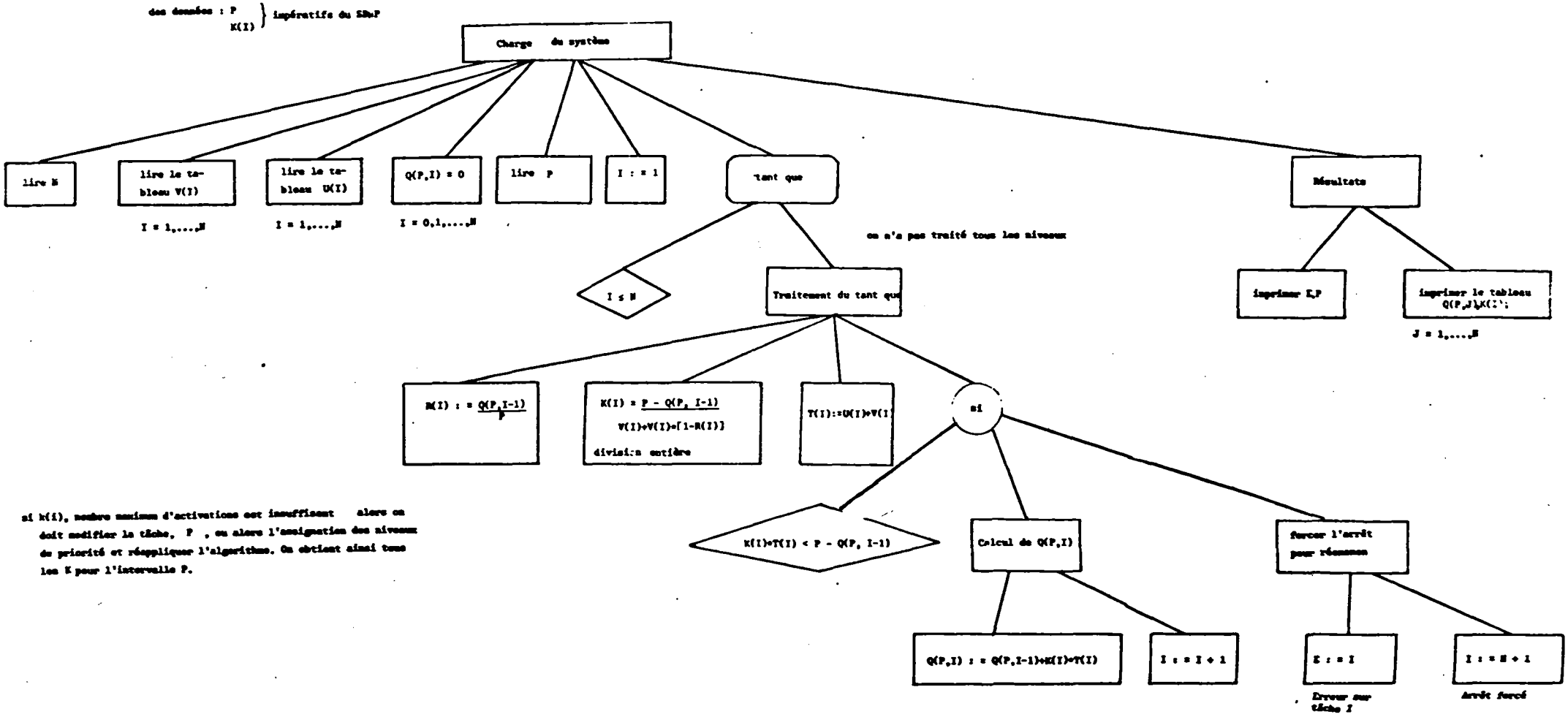
Supposons que notre SBμP soit celui qui est schématisé ci-dessous nous allons construire une primière table où vont apparaitre :

- la période d'activation de chaque tâche (gestionnaire) de périphérique
- la durée du test pour savoir si le périphérique est prêt.

(Résultats tirés de l'expérience des concepteurs [OGD 78]



| tâche | période ms de tests | durée en ms | priorité | |
|---|---|---|---|---|
| scrutation pupitre de commande | 40 | 2 | 2 | |
| lecture convertisseur A/D | 100 | 0,5 | 3 | |
| horloge (lecture) | 15 | 0,1 | 1 | la plus prioritaire |
| écran cathodique | 1000 | 55 | 6 | |
| impression caractère ? | 100 | 0,1 | 7 | la moins prioritaire |
| visualisation analogique | 50 | 0,1 | 4 | |
| afficheurs | 40 | 0,2 | 5 | |




si k(i), nombre maximum d'activations est insuffisant alors on doit modifier la tâche, P , ou alors l'assignation des niveaux de priorité et réappliquer l'algorithme. On obtient ainsi tous les K pour l'intervalle P.

## EXPLICATIONS DE L'ARBRE CHARGE DU SYSTÈME

Les données sont :

le tableau u(i)
le tableau V(i)
la période P
N nombre maximum de niveaux de priorité
la tâche la plus prioritaire correspond à i = 1
au départ de l'algorithme Q(P,0) = 0 ;
Q(P,0) est la tranche de temps pendant laquelle le μP exécute une tâche plus prioritaire (qui n'existe pas en fait).

On fait donc le traitement tant que $i \leq N$ ainsi on traite tous les niveaux de priorité.

R(i) est le temps disponible dans P, pour les tâches plus prioritaires que celle de niveau i

K(i) est le nombre maximum d'activations possibles de ce niveau de priorité pendant P

T(i) est la durée u(i) + V(i) d'une tâche.

Pour tester si le microprocesseur n'est pas surchargé, on vérifie que

$$K(i) * T(i) < P - Q(P,i-1)$$

dans ce cas on ajoute à Q(P,i) la quantité de temps apportée par la nouvelle tâche :

$$Q(P,i) = Q(P,i-1) + k(i) * T(i)$$

si le μP est surchargé, on arrête le traitement.

Exemple fabrique de lard fumé (suite)

Numéro des tâches

| | | |
|---|---|---|
| 1 | terminal 1 | |
| 2 | terminal 2 | x = 1, 2, 3 |
| 3 | terminal 3 | |
| 4 | disquette | |
| 5 | imprimante | |

Contrainte

D(1) = D(2) = D(3) = 5 secondes (temps de réponse) ;
t(1) = t(2) = t(3) = 20 secondes (car trois cochons par minute)
P = 800 s car pour 40 cochons il faut une entête ; nous pouvons prendre les 100 premières secondes.

V(1) = 2,5 = V(2) = V(3) (car vitesse de saisie : 10 c/s ; 25 c/terminal)
V(4) = 0,3 (car 0,15  2 ; lecture et écriture)
V(5) = 5 s ajustement : V(5)  5,50 temps moyen impression ligne car pour l'entête on a en moyenne 18 s : 40 = 0,45 s

| | |
|---|---|
| pour une ligne : | 5,0 s |
| soit : | 5,45 s |

pour P = 100

k(1) = k(2) = k(3) = 1 ; k(4) ≥ 3 ; k(5) = 1

Cas le plus défavorable : pas de recouvrement, les 3 terminaux travaillent et il y a impression de l'entête

Q(p,x) = D(1) + V(1) + D(2) + V(2) + D(3) + V(3) + 18 s
= 3D(1) + 3V(1) + 180
# 0,3  3 + 5  3 + 18 s = 33,9 s ;c'est trop par rapport à 20 s.

Solution avec recouvrement

a) Q(P,x) = 16,5 s (3 lignes ; 5,5 s par ligne)

b) Q(P,x) = 17,4 s (3 lignes ; 5,5 s par ligne ; plus 0,3 s pour disquette)

Ces solutions sont accpetables si on dispose de suffisamment de mémoires tampons.

Si aucun autre périphérique n'oppère a un niveau de priorité supérieur ou égal à n,

Si aucun autre périphérique n'oppère à un niveau de priorité égal à n, on peut déterminer statistiquement la durée cachée sans autre facteur pouvant ralentir le traitement considéré.

Le succès de cette démarche est lié aux deux restrictions suivantes :

1. chaque signal émit par un périphérique ne sera attendu (ou recevable) que par un seul processus

2. un processus (ou tâche) de périphérique ne pourra pas d'elle même invalider les conditions associée au signal émit.

Les résultats obtenus restent une estimation car à priori ont ne connait pas, en langage évolué, la durée des appels de sous-programmes, celles des calculs flottants, etc. Inconnues qui quelquefois peuvent considérablement allonger les temps.

### *7.2.2 Les trois modes d'interaction µP-environnement*

Si le temps réel est plus rapide que le temps propre toute synchronisation est impossible : cela signifie en fait que le µP considéré n'est pas assez rapide pour l'application. Par exemple, il est évident qu'un µP qui a 2 µs de temps d'instruction au minimum ne pourra acquérir une donnée que en gros, toute les

6 à 10 µs. Il ne pourra donc pas traiter une application qui exigerait que l'on échantillonne les donnée à 1 MHz.

Si le temps réel est plus lent que le temps propre, alors l'application est envisageable. D'ailleurs la plus part des processus industriels sont lents (une commutation de vanne demande 10 ms) au regard des µP's (qui dans le même temps réaliseront 1000 opérations élémentaires).

Les temps de mesure des capteurs, les temps de réactions des actionneurs et les temps de conversion AD/DA (analogique-digital et digital-analogique), ralentissent le fonctionnement du µP.

Alors trois solutions de synchronisation permettent au µP de "suivre" l'environnement.

7.2.2 1°) *La boucle de surveillance (périodique)*

Ou le programme boucle sur une série de lectures des différentes grandeurs à acquérir, (valeurs ou états) en ménageant éventuellement des délais pour que les mesures d'une même grandeur se succèdent assez lentement de façon à obtenir une évolution significative.
Dans certains cas la boucle dure assez lontemps et le SBµP n'a pas le temps de réponse assez rapide a certains évènements. On utilisera alors la seconde solution.

7.2.2 2°) *Le SBµP est commandé par évènements*

Certains évènements (fermeture d'un contact, appui sur une touche, passage d'une grandeur par une valeur d'alarme) créent une interruption). Suivant la priorité associée, le P abandonnera la tâche en cours pour prendre un compte et traiter en cours pour prendre en compte et traiter l'interruption.
Il est souvent préférable d'employer ces deux premières méthodes ensembles : dans un fonctionnement de croisière ou utilise la boucle de surveillance de certaines grandeurs, tandis que certains évènements <u>cruciaux</u> sont les seuls capables de créer des interruptions.

7.2.2.3°) Enfin, le coût du matériel décroissant avec le temps, les concepteurs n'hésitent plus a utiliser un matériel plus rapide, au lieu de chercher à utiliser les µP's au maximum de leur possibilités [WIR 77].
On pourra ainsi aller jusqu'à la répartition du travail entre plusieurs µP's (cf : mode C pp 2) de préférence a l'emploi d'un calculateur plus rapide et puissant, mais plus cher.
En résumé, dans la première méthode, l'initiative de l'interaction revient à l'environnement (ou processus à contrôler), tandis qu'à la deuxième méthode, c'est le µP qui est l'initiateur.
Pour ce qui est de la dernière méthode, il s'agit d'un fonctionnement mixte (ou parallèle synchronisé) : le µP (maitre) effectue périodiquement un traitement à partir d'informations quel'environnement doit avoir fourni entre temps (par ADM par exemple).

### 7.2.3 Les difficultées rencontrées si l'on veut adopter le modèle des files d'attentes

Pour décrire brièvement une file d'attente, on utilise généralement la représentation suivante : [LEN 78]

<politique d'arrivée des clients, i.e distribution probabiliste suivie)
<politique de service des clients, i.e distribution prababiliste)
<nombre de serveurs>
(<ordre de service [LIFO, FIFOp avec préemption, FIFO$\bar{p}$ ; tourniquet, échelle des priorité] ; taille de la file d'attente).

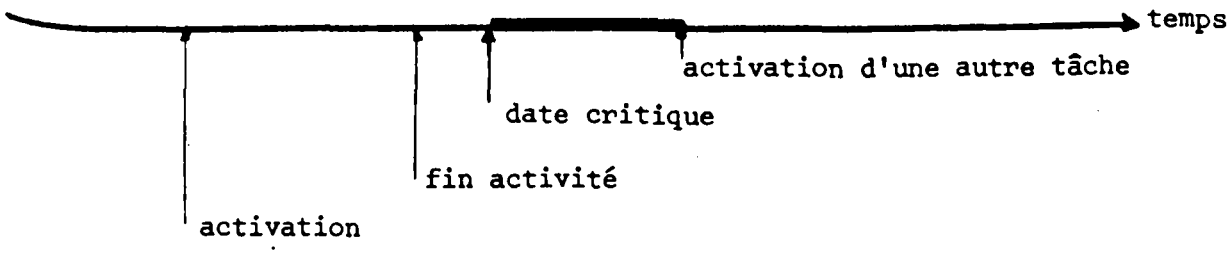
Dans la variété de files d'attente qui ont été étudiées, citons celles qui se rapprochent le plus de notre problème.

1. L'existence de plusieurs classes de clients différents entre elles par :
   - la distribution du temps de service,
   - la priorité,
   - la nature, etc.

Par nature nous voulons dire qu'on peut considérer comme clients, non seulement les tâches, mais aussi les processus intervenant dans ces tâches (cf. chapitre 2).

2. L'impossibilité temporaire du serveur à la fin d'une période de service, d'exécuter traitement, modèle de SKINNER. En effet il n'est pas toujours possible que le μP soit disponible pour servir une interruption. Car dans un programme il peut y avoir des sections critiques où existe une ou plusieurs instructions d'interdiction d'interrompre. Le μP aura alors des périodes d'occupation variables.

3. Le phénomène d'impatience : les clients peuvent se retirer de la file d'attente au bout d'un certain temps. Il faut alors faire apparaître la notion de date critique.




Pour estimer les contraintes de temps, nous avons écarté les modèles probabilistes de la théorie des files d'attente.
La raison (en) est (que) notre problème (SBμP) se situe dans le groupe de ceux qu'on peut qualifier de fixes [GUI 79]. C'est à dire celui des applications pour lesquelles l'utilisateur dira : par exemple :

"le temps de réponse doit être inférieure à D" ;

et non pas :

"le temps de réponse doit être inférieure à D avec la probabilité Pr"

ou alors :

"le temps de réponse moyen doit être inférieur à D.

Dès lors nous ne voyons pas comment déterminer l'une des politiques du modèle. Même avec l'hypothèse restrictive qui veut que dans la thoérie des files d'attente, seul l'environnement peut initialiser les échanges (ce qui n'est pas le cas dans les SBμP).

Tout ce que nous pouvons dire, (et ceci n'avance en rien) c'est que le modèle le plus proche est GI/G/1/(FIFOp ; ∞), qui est en fait le modèle général (G).

Si la mise en oeuvre du modèle s'avère compliquée pour faire des estimations de temps et de taille dans le SB P globalement, elle est très interessante lorsqu'il faut faire des comparaisons précises de techniques d'entrée-sortie.

C'est ainsi qu'il est utilisé avec succès dans [CAM 78] pour comparer l'implémentation des interruptions vectorisées à celle du "polling" (ou interrogation des périphériques par scrutations).

## 7.3 EN QUELS TERMES S'EXPRIMENT LES CONTRAINTES DE TAILLE DU LOGICIEL D'UN SBμP

La longueur N d'un programme est la somme du nombre d'occurrence d'opérateurs (fonctions, procédures) et du nombre des opérandes utilisées [NEM 79] Intuitivement, cette longueur est une fonction $N = f(n_1, n_2, \lambda, s, c)$ où $n_1$ est le nombre d'opérateurs, $n_2$ le nombre d'opérandes (les données), $\lambda$ mesure le langage de programmation utilisé, s mesure le style du programmeur, c mesure la complexité de l'algorithme codé.

$n_1$ Elargissons le nombre d'opérateurs au nombre des tâches admises par le noyau. Il faut donc connaitre non seulement la taille de ce dernier, mais aussi celle des tâches utilisateur et des sous-programmes existant qu'on utiliserait pour écrire moins d'instruction.

$n_2$ Le concepteur est capable de calculer statistiquement le volume des données à stocker dans une RAM.

$\lambda$ Les SBμP nécessitent une programmation des entrées-sorties. Comme ceci ne peut se faire qu'en langage assembleur, un choix du type de codage s'impose. Généralement les programmes en langage évolué sont deux à dix fois plus volumineux que ceux en assembleur ; et l'écriture du programme est plus rapide dans le même rapport.
Cependant lors de l'implémentation réelle d'un programme écrit en langage évolué, le code peut prendre encore plus de place ; car il y a des sous-programmes qui n'apparaissent pas lorsqu'on reste au niveau de la transcription des instructions assembleur/langage évolué.

Avec des frontières floues, nous donnons un tableau qui est basé sur l'expérience des concepteurs analystes et programmeurs. Suivant le nombre d'exemplaire de SBμP à construire, et la vitesse du processus à contrôler, on a :

| facteur de répétitivité / vitesse du processus | prototype et petite série | grande série |
|---|---|---|
| lent | langages évolués | langages évolués/assembleurs (suivant la ROM disponible) * |
| rapide | Assembleurs ou langages évolués | Assembleurs |

* Le fait de rencontrer sur le marché des mémoires de programme de taille fixes données (suivant la configuration de la carte : 1K, 4K, 16K, 32 K bits sur le boitier, avec des extensions possibles jusqu'à 64 KØ) est-il une facilité donnée aux concepteurs ?

Le concepteur peut souhaiter, dans ses programmes, la structuration des données et des traitements ; avoir une écriture (de programme) synthétique lisible et compacte. Il s'en suit alors :

- la souplesse
- des mises au point rapide
- des possibilités d'extension, voire d'indépendance machine et de portabilité

Mac Cabe a proposé une mesure de la complexité des programmes dans le but de détecter les programmes (modules et tâches) qui seront difficiles à tester, ou dont la maintenance sera difficile a assurer. Pour cela il utilise la théorie des graphes, montre comment calculer simplement la complexité d'un programme [MAC 76] (V(G) : le nombre cyclomatique du graphe G associé au programme) et conseille de dimensionner les sous-programme de telle façon que $V(G) \leq 10$.

Ce vocabulaire est moins parlant que celui utilisé par OGDIN : "l'expérience montre qu'un programme de 100 000 bits est le maximum d'information qu'un programmeur peut connaitre parfaitement" en assembleur, du point de vue complexité [OGD 78 ].

Dans le monde des µP's, les programmes sont linéaires. Cette linéarité est conservée même dans les processus temps réels où des traitements, comme les boucles d'attente apparaissent, et le programme principal reboucle sur lui-même.

Les calculs statistiques prenant en compte les résultats des benchmarks réalisés sur microprocesseurs, nous semblent la meilleure base de toute estimation (voir tableaux suivants [PEN 77]).

Pour la taille des tâches : le nombre de leurs instructions étant donné (en langage évolué), on le multiplie par le nombre moyen (d'instructions assembleur générées par une instruction langage évolué). On obtient ainsi une estimation du nombre d'instructions de la tâche en assembleur. La taille est calculée à partir de la longueur moyenne d'une instruction élémentaire en assembleur.

Par exemple : les longueurs des instructions du 6800 A varient entre un et trois octets la longueur moyenne d'une instruction est de deux octets. Elle est de 1,3 pour le 8080 A.

D'ailleurs, les compilateurs déterminent les tailles et durées de chacun des sous-programmes testés, lorsque ces derniers sont parfaitement écrits. [CHO 77].

## CONCLUSION

Avec les algorithmes proposés, il est possible de faire une estimation valable des temps de réponse et de la charge du SB µP. Ceci peut se faire manuellement ou automatiquement.

La taille et les temps (de réponses ou de traitements) étant très liés, l'optimisation de l'ensemble ne peut se faire qu'en même temps ; le programmeur étant celui qui donnera (à la main), la taille minimale à son programme. En effet il sait quelle variable est utilisée par telle ou autre tâche, quand il

## CARACTÉRISTIQUES DES µP'S TESTÉS

| Process technology | A.l.u. size (bits) | Memory width (bits) | Instruction length (bits) | Number of instructions | Stack | Registers | Instruction cycle time (µs) | Add time (µs) | Interrupts |
|---|---|---|---|---|---|---|---|---|---|
| p-m.o.s. | 4 | 8 | 4, 8, 12 | 60 | 7 × 12 | 24 × 4 + Acc | 10·8 | 10·8 | 1 |
| n-m.o.s. | 8 | 8 | 8, 16 | 48 | none | 64 × 8 + Acc | 2 | 2–5 | many |
| n-m.o.s. | 8 | 8 | 8, 16, 24 | 78 | external | 6 × 8 + Acc | 2 | 2 | 1† |
| n-m.o.s. | 8 | 8 | 8, 16, 24 | 72 | external | Index (1 × 16) + Acc (2 × 8) | 1 | 4 | 1 |
| p-m.o.s. | 8 | 8 | 8, 16 | 46 | external | 3 × 16 Base + Acc + Extension | 2 | 38 | 1 |
| c-m.o.s. | 12 | 12 | 12 | 40+ | none | Acc (1 × 12) + MQ (1 × 12) | 0·5 | 2·5 | 1† |
| n-m.o.s. | 16‡ | 10 or 16 | 10, 20, 30 | 87 | external | 6 × 16 | 0·4 | 4 | 1† |
| p-m.o.s. | 16 | 16 | 16 | 45 | 10 × 16 | 4 × 16 | 2 | 8 | 6 |
| n-m.o.s. | 16 | 16 | 16, 32 | 69 | external | 16 × 16 (multiple copies) | 0·33 | 7 | 16 |

† Extendable to multi-level with external logic. ‡ Organized internally as byte serial.

## RÉSULTATS DE BENCHMARKS

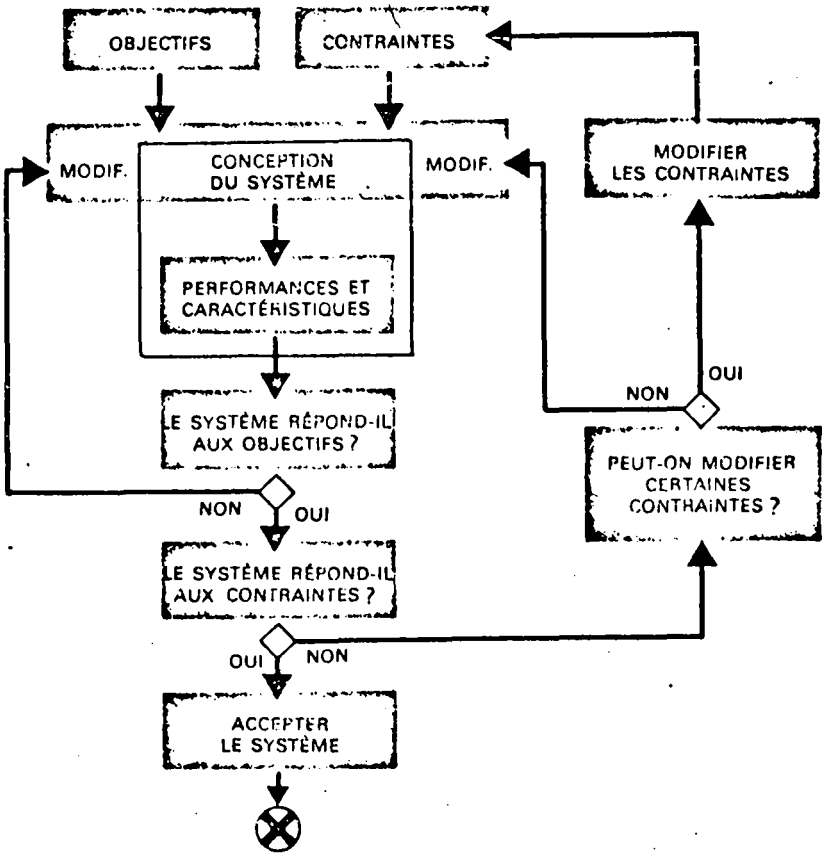
| | 8-Bit Addition | | | 16-Bit Addition | | | 8-Bit Multiplication | | | Output (Programmed) | | | Output (Interrupt) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N.S. | Bytes | Time (µs) | N.S. | Bytes | Time (µs) | N.S. | Bytes | Time (µs) | N.S. | Bytes | Time (µs) | N.S. | Bytes | Time (µs) |
| | 14 | 19 | 324 | 14 | 19 | 561 | — | — | — | 14 | 19 | 124[a] | 26 | 32 | 294[a] |
| | 6 | 12 | 51 | 20 | 27 | 94 | 40 | 49 | 514 | 8 | 10 | 40 | 19 | 23 | 87 |
| | 4 | 10 | 28 | 5 | 11 | 38 | 20 | 35 | 374 | 7 | 14 | 32 | 17 | 25 | 102 |
| | 3 | 9 | 13 | 6 | 18 | 26 | 12 | 24 | 206 | 10 | 21 | 41 | 33 | 53 | 98 |
| | 4 | 7 | 120 | 7 | 13 | 230 | 27 | 40 | 2306 | 6 | 12 | 188 | 56 | 64 | 1060 |
| | 4 | 4[b] | 10 | 9 | 9[a] | 23 | 44 | 44[b] | 500 | 7 | 7[b] | 22[a] | 10 | 10 | 35[b] |
| | 8 | 28 | 28 | 3 | 12 | 12 | 18 | 52 | 182 | 5 | 16 | 16[d] | 23 | 74 | 95[d] |
| | 3 | 6 | 24 | 3 | 6 | 24 | 15 | 30 | 546 | 7 | 14 | 62[e] | 14 | 28 | 136[e] |
| | 3 | 12 | 22 | 3 | 12 | 22 | 3 | 12 | 39 | 5 | 4 | 39[f] | 4 | 10 | 29[f] |
| | 4·25 | 9·5 | 53 | 9·5 | 17·25 | 97 | 25 | 37 | 850 | 7·75 | 14·25 | 75·25 | 29 | 41·25 | 334 |
| | 4·66 | 15·3 | 24·66 | 3 | 10 | 19·33 | 12 | 31·33 | 255 | 5·66 | 11·33 | 39 | 17 | 37·33 | 87 |

N.S. = number of program statements.

(a) 4-bit characters.
(b) 12-bit characters.
(c) 24-bit addition.
(d) 16-bit words.
(e) either 8- or 16-bit units, wired option.
(f) either 8- or 16-bit units, program option.

faut la conserver ou libérer l'emplacement et les endroits de programme qu'elle occupe.

En s'aidant des Benchmarks et autres résultats de compilateurs, il peut avoir une idées de la taille de ROM nécessaire. Le processus de conception étant le suivant :



Pour ce qui est de la programmation en langages évolués, et sans entrer dans une discution comparaison détaillée avec les langages assembleurs, leur choix s'adapte à l'utilisateur de COSMIC. En effet, ils mettent à la disposition de "personnel n'ayant pas de qualification informatique spéciale, mais connaissant très bien le processus et ses impératifs" un outil qui permet la communicabilité entre l'usager, le processus et mène le concepteur [TOU 75] .
Les concepteurs de SB P affirment que les programmes suivraient (nous ne l'avons pas vérifié) la règle des 80/20. Qui veut que 80 % du temps CPU soit consommé par l'exécution de 20 % de code. Le tout est de trouver ces 20 % pour tenir les contraintes de temps (boucles, périphériques).

Lorsque la faisabilité de l'application est mise en cause, on peut faire :

a) la modification des spécifications fonctionnelles, la division en tâches des sous-problèmes pour faire des compromis entre ce qui peut être fait par le hardware et ce que logiciel peut traiter.

- les équipements rapides (cablés ou programmes) pouvant traiter les problèmes simples
- pendant que le µP assure la gestion des fonctions moins critiques pour les temps

b) l'utilisation de familles logiques adaptées est un moyen matériel. Ainsi dès le début de choix des équipements, on peut introduire la logique rapide dans les parties critiques de l'application seulement. Ce type de souci amène le concepteur à examiner et comparer les différentes technologies du matériel.

c) Enfin il existe d'autres moyens matériels, qui peuvent contribuer aux respect des contraintes. Ils sont trop fins et liés à la circuiterie pour que nous les abordions ici.

# BIBLIOGRAPHIE


[BAT 79] Roger R. BATE et D.S. JOHNSON
*Pascal software supports real time multiprogramming on small systems*
Electronics, 7 june 1979

[CAM 78] J.M. CAMPENHOUT and Paul NOTREDAME
*A stochastic model for closed-loop preemptive microprocessor I/O organisations*
IEEE transactions on Computers, Vol C-27, n° 12, Déc 1978

[CAN 79] Pat CANDILL
*Using assembly coding to optimize high level language programs*
Electronic February 1979

[CHO 77] *Execution time Analysis for Real Time µC programs*
University of California, Berkeley, 1977

[CUN 69] A. CUNNIGTON
*Contraints of Real Time computer systems*
Ericsson Technico, VoL 25, Sweden (1969), pp. 50-90

[GIR 79] D. GIROD et R. DUBOIS
*Au coeur des micorprocesseurs*
Ed. EYROLLES 1979

[GRO 79] C. GROSS
*Un micro pour votre automatisme. Oui, mais attention au cahier des charges*
A21 n°75, mars 1979

[MAC 76] T.J. MAC CABE
*A complexity measure*
IEEE transactions on Software Engineering, vol SE2, n° 4, pp 308.320,
déc. 1976

[MEM 79] Gérard MEMMI (ECA Automation)
*Mesures quantitatives et qualités du logiciel*
Journée Génie logiciel, pp 247-256

[PEN 77] B. K. PENNEY
*The implementations of µP architecture on speed programming and memory size*
The Radio and Electronic Engineer, Vol 47, n° 47, pp. 522-528, nov 19

[TOU 75] J.M. TOULOTTE
*Dispositifs de commande en temps réel*
Dunod Université 1975

[WIR 77] M. WIRTH
*Toward a discipline of real time programming*
CACM, août 1977, n° 8, Vol 20

# PLAN DU CHAPITRE 8

## INTRODUCTION

COSMIC est un ensemble de modules intégrés dont la mise en oeuvre permet de construire un SBµP, ne serait ce que sur le papier.

Dans ce qui suit nous donnons l'organisation générale de COSMIC, en rappelant les différents modules qui le composent, leurs liens et rôles.

Ceci nous permettra de définir le cadre et la configuration de COSMIC, et d'indiquer comment nous pourrions le compléter.

Ensuite nous justifierons le choix d'un langage de programmation comme langage de définition ; et pourquoi PASCAL, parmi tant d'autres langages programmations.

## 8.1 ORGANISATION GENERALE DE COSMIC

Notre démarche pour la définition de COSMIC a été la suivante : nous avons défini un noyau logiciel auquel s'ajoutent des structures de données et de contrôle correspondants aux tâches décrites par le concepteur. Ce noyau logiciel est une structure d'accueil de la description faite par le concepteur, lorsqu'elle n'existe pas.

On l'appelle plus couramment MONITEUR TEMPS REEL et il faut, examiner ceux qui existent, connaître comment sont activées les tâches, le nombre maximum des tâches qu'ils admettent, et la taille de ces moniteurs.

Des aides au niveau de choix de l'équipement microprocesseur de base, estimation des temps d'exécution et des tailles mémoires, font que COSMIC s'apparente aussi à un système de mise au point et de développement.

### 8.1.1 Les fichiers et modules de COSMIC

Pour la description de COSMIC nous avons parlés d'abord des différents ensembles de fichiers qui sont utilisés par les modules et qui font partie de la base de données de COSMIC. Ce sont :

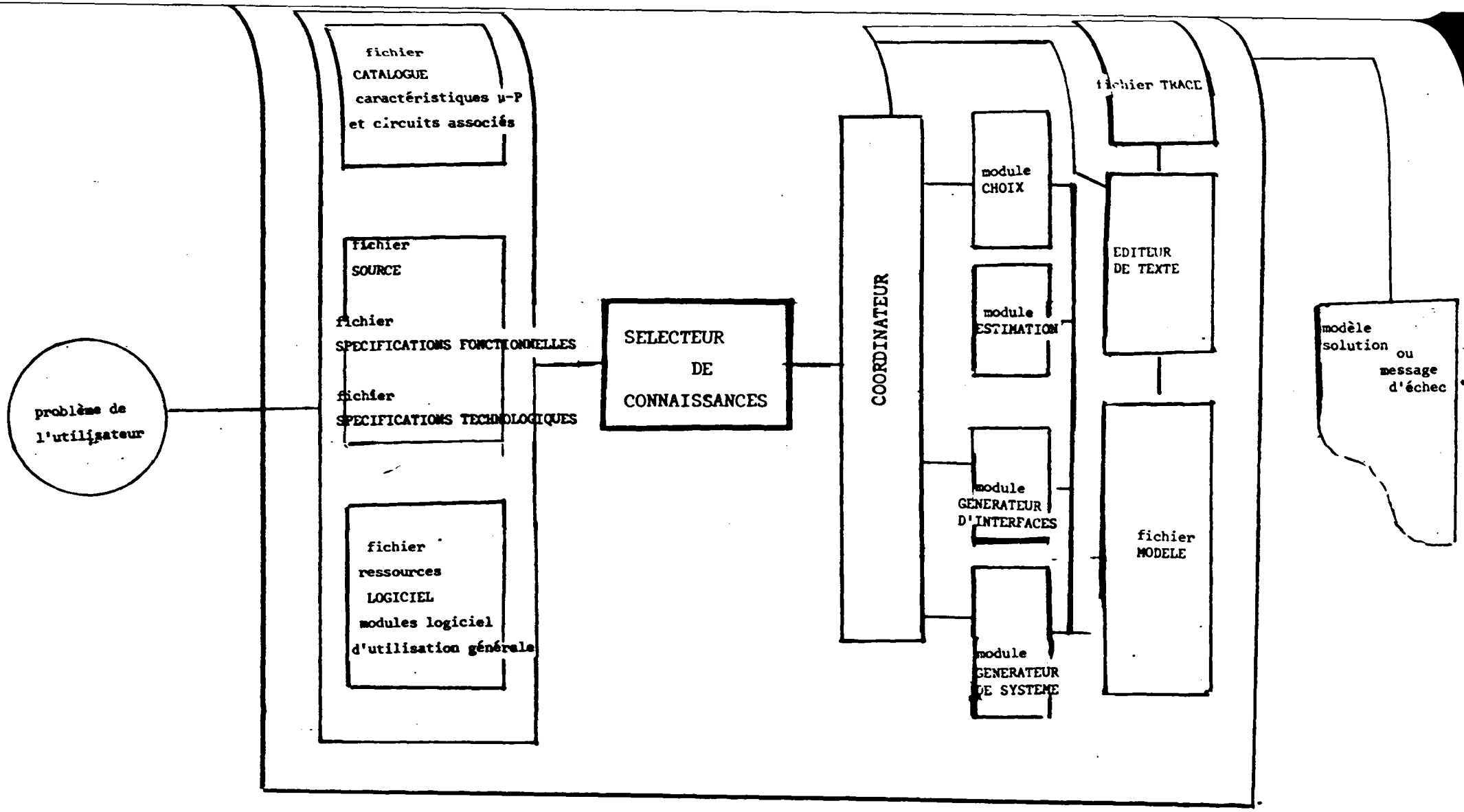
LES FICHIERS

1) Le fichier CATALOGUE

C'est une base de données technologiques qui contient les caractéristiques des équipements à base de microprocesseur et les circuits associés.

2) Le fichier SOURCE contient une image de la description du problème faite par le concepteur. Ainsi on y trouvera toujours l'image actualisée de la description du problème.

Il est divisé en deux fichiers appelés fichier SPECIFICATIONS FONCTIONNELLES et fichier SPECIFICATIONS TECHNOLOGIQUES.




SCHEMA SYNOPTIQUE DE COSMIC.

S'y ajoutent deux fichiers résultats du travail de conception :

3) fichier TRACE qui contient une trace de la résolution du problème. Le fichier montre les étapes qui matérialisent la recherche d'une solution par COSMIC.

4) fichier MODELE contient une description fonctionnelle de la solution. Lorsque COSMIC ne peut pas déterminer une solution au problème de l'utilisateur, un message d'échec est écrit.

5) Enfin la BIBLIOTHEQUE : Elle est constituée par un ensemble de ressources logicielles, i.e. des modules logiciels d'utilisations générales.

LES MODULES

1) Le module GENERATEUR DE SYSTEME construit, à partir des modules qui composent le noyau logiciel et de la description du problème, le système utilisateur.

2) Module GENERATEUR D'INTERFACES, assigne à partir des spécifications technologiques et du matériel microprocesseur choisi, des rôles aux ports d'E/S des circuits d'interface programmes.

3) COORDINATEUR : ce module constitue le superviseur du système COSMIC. Il réalise essentiellement la fonction d'interprétation des commandes émises par les autres modules ou par le concepteur à partir du poste de travail.

4) Module CHOIX : Il permet le choix des différents boîtiers constituants le SBµP, à partir du fichier CATALOGUE et des spécifications indiquées par le concepteur (fichier SOURCE). Nous y regroupons :
le choix du microprocesseur, l'examen des standards et des circuits dédiés, le choix des mémoires et des interfaces, etc...

5) Module ESTIMATION

Il permet au concepteur, qui connait le langage utilisé sur la machine cible et les périphériques du SBµP, de faire une estimation des temps de traitements, la taille des programmes et des données.

Dès que la durée des tâches est déterminée, on peut calculer la charge du système et faire une estimation des temps de réponses.

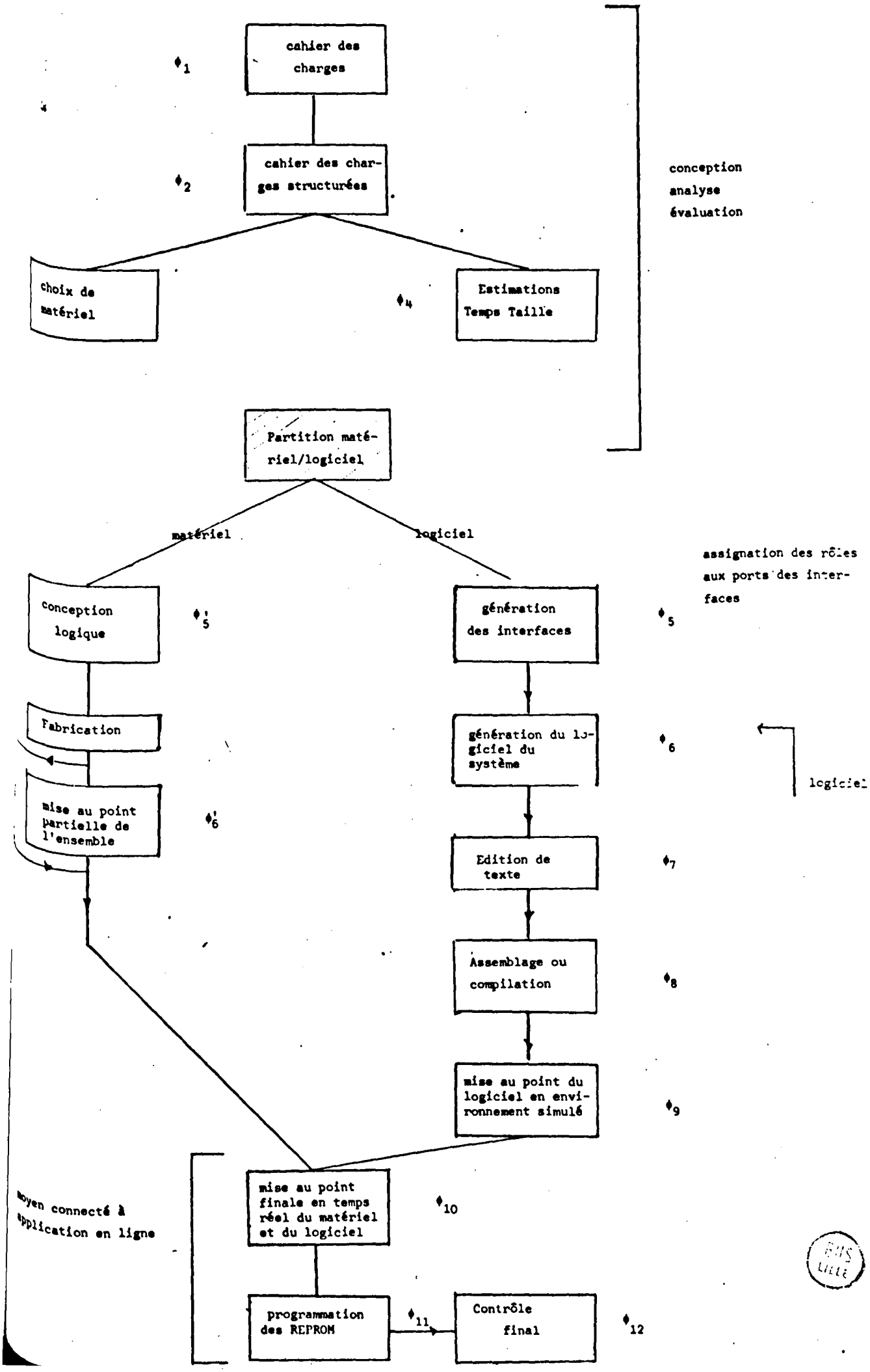
### 8.1.2 Principe de fonctionnement général de COSMIC

Le schéma de la page suivante représente le déroulement des différentes opérations dans le cas le plus simple (conception séquentielle, utilisateur unique). L'enchainement de ces opérations est assuré par le langage de commandes .

Le concepteur définit grâce au langage de définitions les tâches de son application. Ces tâches sont introduites en mode traitement par lots.

La méthode pour écrire une spécification est itérative et de nature descendante. L'utilisateur commence par relire son cahier des charges, plusieurs fois. Ainsi il pourra obtenir les types des données du problème, quelques relations et les assertions qui expriment les lois logiques auxquelles doit répondre le SBµP.

Une étape de conception sera la réalisation d'une phase $\phi$. Nous en énnumérons 12.

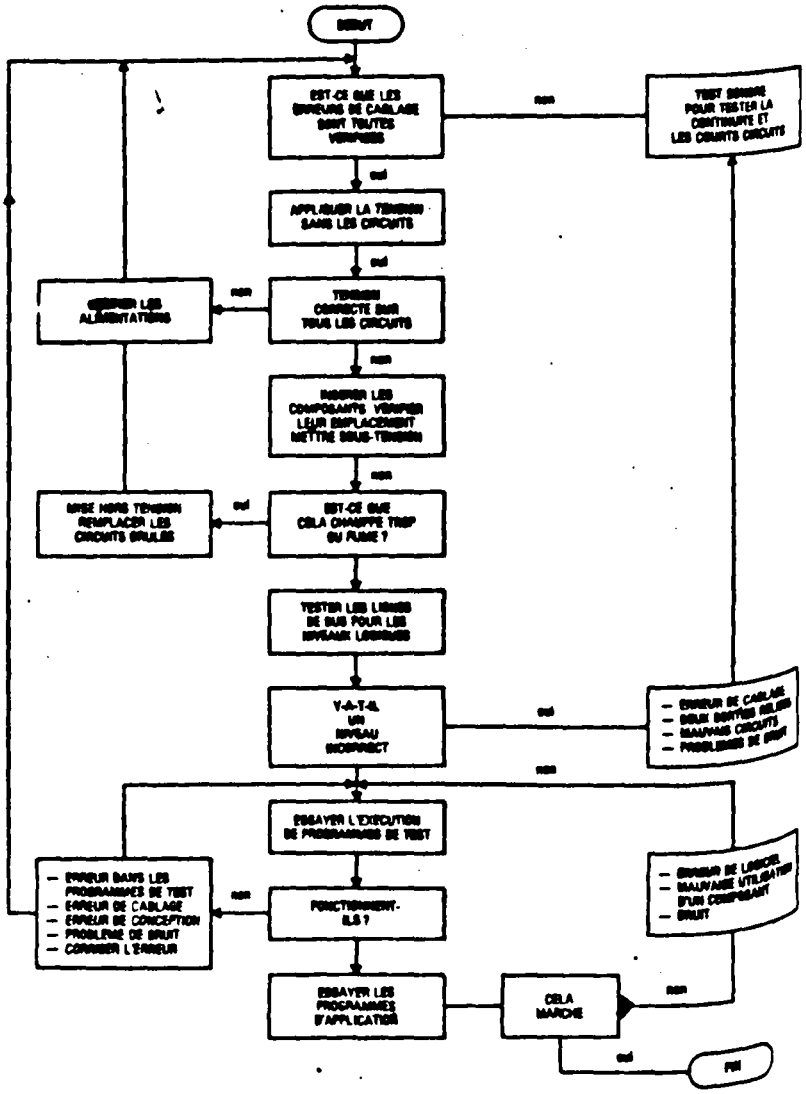

cahier des charges
$\phi_1$
cahier des charges structurées
$\phi_2$
choix de matériel
$\phi_4$
Estimations Temps Taille
conception
analyse
évaluation
Partition matériel/logiciel
matériel
logiciel
assignation des rôles aux ports des interfaces
Conception logique
$\phi'_5$
génération des interfaces
$\phi_5$
Fabrication
mise au point partielle de l'ensemble
$\phi'_6$
génération du logiciel du système
$\phi_6$
logiciel
Edition de texte
$\phi_7$
Assemblage ou compilation
$\phi_8$
mise au point du logiciel en environnement simulé
$\phi_9$
moyen connecté à l'application en ligne
mise au point finale en temps réel du matériel et du logiciel
$\phi_{10}$
programmation des REPROM
$\phi_{11}$
Contrôle final
$\phi_{12}$

Nous ne nous sommes pas attardés sur l'aspect matériel qui, quoique déterminant, peut être difficilement abordable par un néophite.

Les phases $\phi'_6$ et $\phi'_{10}$ peuvent se résumer par l'organigramme suivant :

$\phi'_6$

$\phi'_{10}$




**- Organigramme de mise au point**

### 8.1.3 Le cadre de COSMIC

Ce cadre se justifie après la comparaison des différents outils de développement existants.

En effet, il y a globalement deux types d'outils de développement de logiciels pour µP's. Les outils spécifiques et les outils généraux.

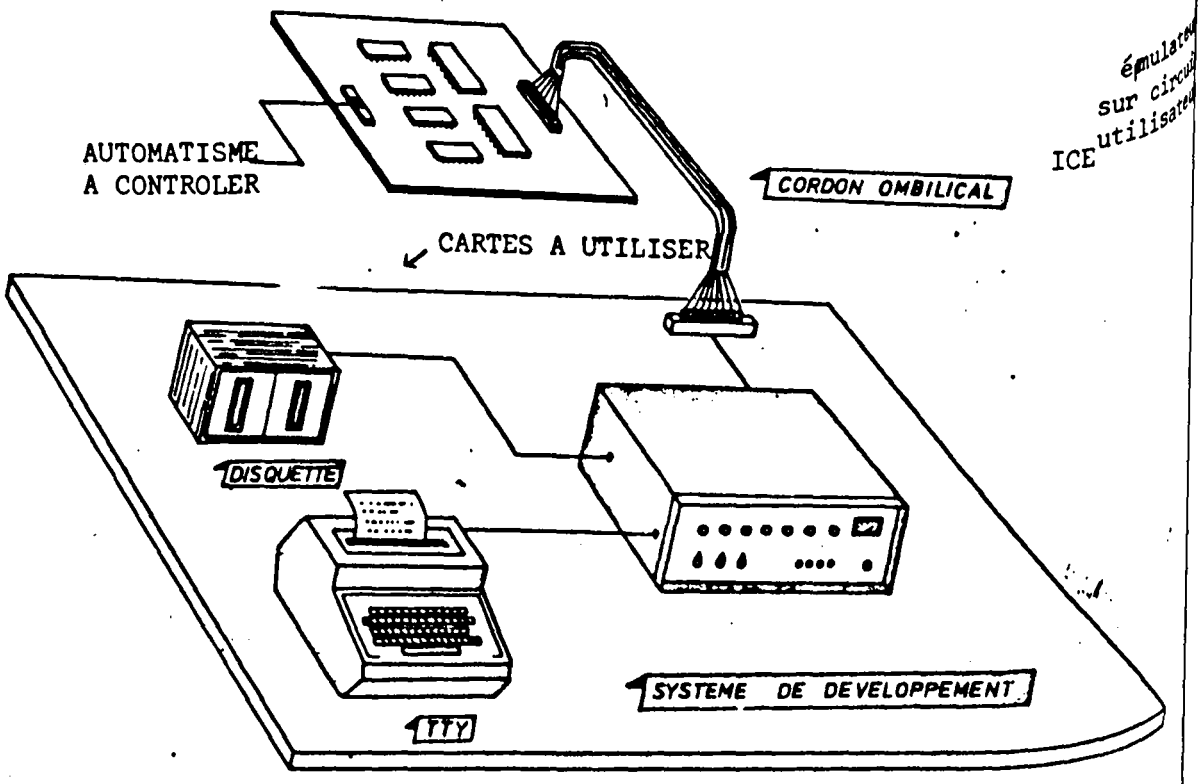
| Caractéristiques | Outils spécifiques | Outils généraux | |
|---|---|---|---|
| généralités | à base de µP mono-poste pour développement d'un µP. ex. : MDS, EXORCISER, FORMULATOR, etc. | mini-ordinateur mono ou multi-postes logiciels CROISES | gros système, réseaux temps partagés |
| Production du code | Très correctement lents | parfaitement | parfaitement, pas optimisé rapidement. |
| mise au point | Simultanée, en quasi temps réel du matériel et du logiciel | petit moniteur sur carte PROM accessible par connexion µP | Simulation d'exécution sur µP (pas en temps réel), mise au point partielle, pas de mise au point intéractive. |
| Avantages majeurs | finesse de la mise au point autonomie complète | | . pas d'investissements initials . possibilité de changer facilement de µP. . gestion des modules simples . multipostes |
| Phases traitées | $\phi_7$ $\phi_8$ $\phi_9$ $\phi_{10}$ $\phi_{17}$ | $\phi_7$ $\phi_8$ $\phi_9$ | $\phi_7$ $\phi_8$ $\phi_9$ |

Ces outils contribuent chacun et de manière plus ou moins complète à toutes les phases du processus de conception schématisé.

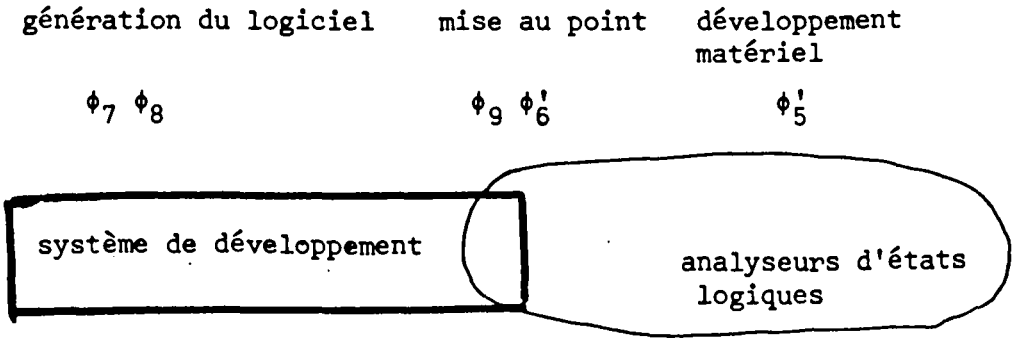
Les outils spécifiques dont nous parlons ici sont :

Dès µθ dotés de facultés supplémentaires d'aide à la mise au point matérielle, grâce à des modules de simulation en temps réel du µP. Ces derniers sont généralement désignés sous le nom de modules 1n Circuit Emulator. Le µP situé sur la carte est retiré de son support et remplacé par un cordon ombilical relié au module ICE. Ce dernier contient un programme spécialisé, qui fournit à la carte application, par l'intermédiaire du cordon ombilical, des signaux identiques à ceux du µP.

Il est alors possible d'arrêter le programme, de l'exécuter pas à pas, d'examiner ou de modifier le contenu d'un registre, d'enregistrer en temps réel une trace de l'état du système pendant un intervalle $[t_1, t_2]$ de temps donné pour le visualiser en différe...



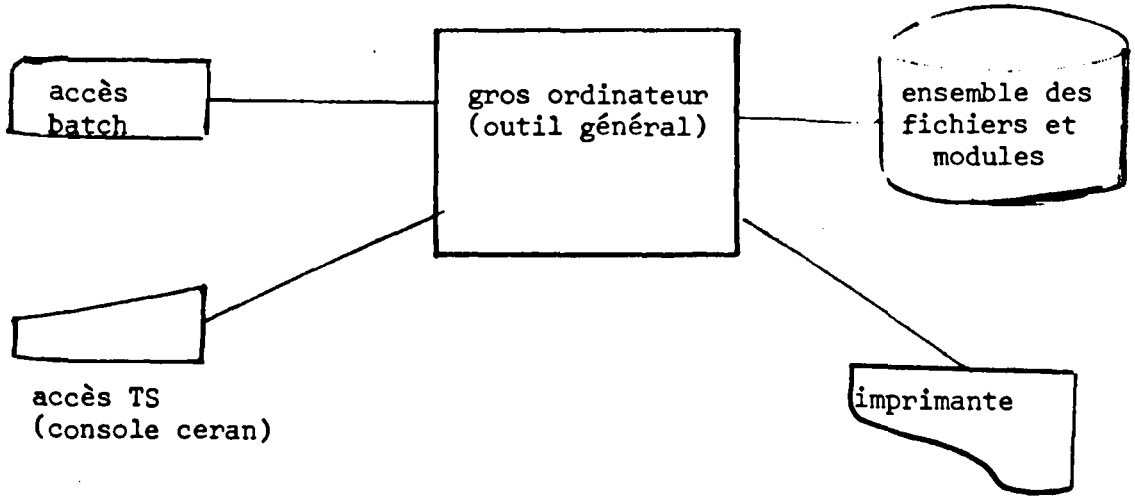
Il faut cependant que nous parlions des analyseurs d'états logiques, qu'on peut aussi classer parmi les outils spécifiques. Ils interviennent dans les phases $\phi'_6$ et $\phi'_{10}$ et n'ont pas d'intersection vide avec les systèmes de développements comme le montre le schéma ci-dessous.




Là encore il y a lieu de faire un choix :

système de développement suffisant ? sinon nécessité d'un analyseur. En faisant attention que ce dernier fassent bien, au niveau du matériel, ce que le système de développement ne peut faire.

COSMIC se situe dans le cadre des outils généraux



Deux types de modules pourraient trouver leur place dans COSMIC : les MODULES EDITION DES IMAGES MEMOIRES et le MODULE MESURE QUANTITATIVE ET QUALITATIVE DU LOGICIEL.

1) Le premier permet la lecture et l'enregistrement des images mémoire (positions) sur un support, ainsi que d'autres informations complémentaires.

Pour le construire, on introduit des sous-programmes-sondes dans les parties du programmes principal, que nous appelerons moniteurs logiciels [EBO 76]. Ils sont conçus de telle manière que :

. leur durée d'exécution soit constante,
. qu'ils ne s'exécutent pas de manière aléatoire
. qu'ils appartiennent au 80 % des routines qui, même si elles occupent assez de place mémoire, ont une durée d'exécution courte ;

Ceci pour ne pas perturber considérablement le déroulement des programmes d'application.

Ce types d'outils, souvent donnés par les constructeurs de matériel de SBµP, permet des autotests des mémoires. Quelquefois on les trouve dans la BIBLIOTHEQUE des utilitaires.

Module mesure quantitative et qualitative du logiciel

JUSTIFICATIONS : Dans plusieurs endroits de cette dissertation, nous avons suggéré : la décomposition du programme d'applications en tâches. En prenant en compte l'importance des facteurs taille et temps nous avons conseillé au concepteur de faire en sorte que :

. les tâches les plus prioritaires soient les plus courtes
. les programmes d'interruption aient une longueur très limitée.

Ce qui manque un peu de précision car il faut pouvoir donner une mesure exacte (si possible) sur ces programmes.

INTERET : Les seuls résultats connus sont issus de l'expérience, et les recherches permettent de lier la conception intuitive de la complexité d'un programme à des estimateurs.

L'intérêt du module complexité serait d'en énumérer celle d'un programme et d'estimer le nombre d'erreurs probables avant la période de tests, dès la compilation du programme.

Ce serait là une continuation possible de notre travail.

## 8.2 LE LANGAGE DE DEFINITION

Par langage de définition, nous désignons un langage qui permettrait aussi bien la spécification que la mise en oeuvre du problème.

Le concept de langage de spécification correspond à la recherche d'une méthode pour poser le problème avant de la résoudre.

Or nous souhaiterions que notre langage permette non seulement la saisie des besoins de l'utilisateur, leur contrôle, la maintenance ou la documentation mais aussi la mise en oeuvre sur un microprocesseur.

Un langage de programmation peut-il jouer le rôle d'un langage de spécification ? Quels sont les atouts de PASCAL à ce niveau ?

### 8.2.1 Pourquoi un langage de programmation

Pour justifier le choix de tel ou autre langage de spécification, nous utilisons les critères d'évaluations des langages énumérés dans [DEM 79]. (voir tableau ci-après). Pour quelques-uns de ces critères, nous faisons ci-dessus un commentaire - justification.

a) l'aspect statique/dynamique (du langage)

Notre langage doit non seulement décrire le problème (aspect statique), mais aussi permettre d'indiquer le processus (particularité des SBμP).

b) Dispositifs automatiques

Le langage prévu dans COSMIC doit aussi bien permettre des spécifications "sur papier" qu'il donnerait lieu à un traitement automatique, en particulier à des fins de documentation : (définition de l'application, spécifications technologiques).

c) Le cadre linguistique

Nous avons le choix d'exprimer les spécifications en langage naturel, ou alors sous forme d'automates, de tables, de diagrammes, Réseaux de Pétri, diagrammes de transitions, etc...

Nous avons choisit le formalisme qui soit le plus près possible du langage naturel [OGD 78] car les utilisateurs sont des non-informaticiens.

d) Ce problèmes des utilisateurs est aussi lié au niveau d'abstraction et à la facilité d'apprentissage du langage. Pour tous les formalismes existants, on demande un minimum d'effort à l'utilisateur. Il n'y a que le langage Z qui ne demande pas aux utilisateurs d'avoir des notions de programmation. Le niveau d'abstraction souhaité ici est celui qui accepte que l'utilisateur puisse oublier certains détails et y revenir. Il peut aussi faire abstraction des techniques d'implémentation qui pourront être ultérieurement utilisées.

e) La description de la syntaxe et de la sémantique :

Le formalisme à utiliser doit permettre de décrire la structure du système, les relations entre ses éléments (syntaxe), ainsi que le rôle et la signification de ces éléments (sémantique).

f) Souvent on reproche aux langages de programmation de n'être pas adaptés à la spécification car :

| Formalismes / Critères | Langages de programmation | Tables de décision | Diagrammes de transition | Réseaux de Petri | HIPO | Warnier, Jackson | SREM/SREP | ISDOS | SADT | Types abstraits de données | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Statique ou dynamique | Dynamique | Habituellement dynamique, mais peut être statique | Dynamique | Dynamique | Dynamique | Dynamique | ? | Plutôt dynamique | Statique | Statique | Statique |
| Niveau d'abstraction | Des facilités d'abstraction existent, mais tous les détails doivent finalement être donnés | Bas | Bas | Bas | Bas (dépend de l'utilisation) | Des facilités d'abstraction existent, mais tous les détails doivent finalement être donnés | Haut(?) | Haut | Haut | Haut | Haut |
| Généralité (c'est-à-dire portée des domaines d'application possibles) | Besoins du langage | Général | Général | Systèmes parallèles et temps-réel | Plutôt orienté vers les problèmes de gestion | Réputé pour être général mais très orienté vers les problèmes de gestion | Général plutôt orienté vers les problèmes temps-réel | Orienté vers les problèmes de gestion | Général ? | Général | Général |
| Aides dans le processus de décomposition ? | Oui, au moins dans les bons langages | Non | Non | Non | Non | Oui | Oui | Oui | Oui | Oui | Oui |
| Existence d'un (bon) formalisme de spécification et de conception? | Normalement non (des efforts récents vont dans cette direction) | Non | Non | Non | mieux que rien | Oui | Oui | Oui | Oui | Oui | Oui |
| Une théorie saine et heureuse? | En partie pour quelques langages | Oui | Peut être définie | Oui | Non | Non | ? | Non | Non | Oui (algèbres initiaux) | Oui |
| Cadre linguistique | - | Tables | Diagrammes (ou automates) | Diagrammes (ou automates) | Langage naturel | Arbres (plus quelques constructions des langages de programmation) | Diagrammes plus quelques constructions des langages de programmation | Peu clair | Diagrammes | Théorie des ensembles, calcul des prédicats | Théorie des ensembles, calcul des prédicats, syntaxe type-ALGOL |
| Description de la syntaxe, de la sémantique, ou les deux ? | les deux | les deux | Syntaxe | Syntaxe | les deux | les deux | les deux (?) | Syntaxe | Syntaxe | les deux | les deux |
| Support graphique ? | Non | Non | Oui | Oui | Non | Arbres | Oui | Non(?) | Oui | Non | Non |
| Utilisateurs | Programmeurs | Utilisateurs, analystes, programmeurs | Utilisateurs, analystes, programmeurs | Analystes | Analystes | Analystes, Programmeurs | ? | Analystes | Analystes | Analystes Programmeurs | Utilisateurs (?!) analystes, programmeurs. |
| Facilité d'enseignement | Facile | Facile | Facile | Moyennement difficile | Facile | Plutôt facile | ? | ? | ? | Difficile | Vraiment difficile |

$f_1$) Il nécessitent un trop grand niveau de détail.

$f_2$) Ils sont essentiellement dynamiques (aspects procédural des langages de programmation usuels). En effet ils obligent à décrire COMMENT sont calculés les résultats, alors que l'utilisateur à tendance à ne dire que ce qu'on calcule. Il répond à la question : QUE CALCULE-T-ON ?

Pour le premier reproche : le détail est nécessaire tout-de-suite (cf. la longueur de mot) pour un SBµP. En ce qui concerne le second "défaut" des langages de programmation, on peut l'éviter par l'utilisation des instructions d'appels de sous-programme. On donne ainsi un nom à un sous-programme en négligeant temporairement son contenu. (compilations séparées possibles en PASCAL 80).

### 8.2.2 Pourquoi PASCAL

Il y a plusieurs raisons qui plaident en faveur de PASCAL et beaucoups d'auteurs on fait cette justification de succès. Il n'est pas un langage de spécification (statique) mais il est définit de façon précise et non ambiguë (base théorique).

. Les langages de spécification sont au stade des études ; même le plus prométeur d'entre-eux : Z n'a pas encore été expérimenté pour les µP's. Citons les autres RSL, PSL, SADT...). L'étude d'un nouveau langage doit s'appuyer sur ceux existants.

En résumé, PASCAL permet la structuration des programmes (données et procédures), une lisibilité accrue et une sécurité d'emploi. [RAV 79]. En plus son apprentissage est favorisée par son implantation sur diverses machines et une large diffusion d'ores et déjà observée au niveau des microprdinateurs du commerce.

### 8.2.3 Le langage de commande

Un SBµP peut être réalisé par une équipe et non par une personne.

Dans certaines équipes interviennent d'ailleurs des utilisateurs.

Certaines phases peuvent être traitées en parallèles, comme $\phi_3$ et $\phi_4$. D'ailleurs, l'un des fichiers (au moins), le fichier CATALOGUE peut être utilisé par un utilisateur de LEDA [PLE 79].

Il est nécessaire de donner à chaque personne utilisant COSMIC la possibilité de s'identifier, d'appeler tel module, de modifier les contenus de certains fichiers et d'obtenir selon ses besoins un état de sortie, etc.

Ce langage sera donc défini comme un automate d'états finis, composé de plusieurs commandes de changements d'états, comme c'est le cas pour LAPAGE [CAV 79].

Remarque : Nous avons pensé à la possibilité de donner au langage de définition des capacités voisines de celles des CHDL "Computer Hadvare Design Langages". Ces langages décrivent les systèmes matériels sur trois niveaux :

Description algorithmique

Le comportement du système est spécifié en termes de relations entre les variables d'entrée et de sortie.

Description fonctionnelle

Le système est décrit comme un algorithme.
Les opérateurs peuvent (ou non) correspondre à son implémentation matérielle réelle.

Description structurelle

Description de la réalisation matérielle.
Les opérateurs sont des éléments matériels, les variables des éléments de mémorisation.

Les deux types de langages CHDL qu'on rencontre sont :

a) Ceux où l'algorithme est décrit par une séquence d'instructions avec des possibilités de contrôle du déroulement par des instructions conditionnelles : type procédural.

b) Dans le type non procédural, l'ordre d'écriture des instructions n'est pas significatif. Le déroulement des opérations dépend des tests "permanents" sur les variables d'état, associées aux instructions.

Nous avons laissé de côté cette dernière partie car elle est :

. trop liée au matériel (niveau très fin) ;

. l'utilisateur ne peut l'aborder. Seul un automaticien-informaticien peut utiliser une approche de ce type ;

. c'est l'objectif du langage LEDA [PLE 78], et du système MAS [Zac 77].

## CONCLUSION

Ce qu'est COSMIC

il ne prétend pas :

. automatiser toutes les phases de conception, aspects logiciels ou pas, d'un SBµP comme ISDOS [WAT 76], et MAO [TEI 71].

. donner des recettes utilisables quelque soit le SBµP

. être un outil nouveau. Il se veut un complément aux outils existants, qui sont souvent destinés aux informaticiens avertis.

Ainsi il prétend

. apporter une solution aux non informaticiens qui n'ont pas le savoir faire et l'expérience des concepteurs. A des questions de type : comment allouez-vous des rôles aux ports ?
comment déterminez-vous la taille nécessaire ?
comment déterminez-vous les durées ?
le concepteur même de bonne fois n'a pas de recette toutes établies.

. accélérer ou simplifier (et non automatiser) ce qui peut l'être.

. enfin permettre à l'utilisateur de ne pas se perdre dans le labirynthe du monde des µP's (en constante évolution).

# BIBLIOGRAPHIE

[ANT 77] J.O. ANTON ; C.Y. CHRISMENT ; J.B. CRAMPES ; M.F. DEBAISIEUX ; J.H. LUGUET.
*"Principles of computer-assisted conception for automatic elaboration of computerized information system".*
IFIP 77.

[CAV 79] J.L. CAVARERO.
*"LAPAGE : un modèle et un outil d'aide à la conception des systèmes d'information".*
Thèse d'état, NICE Juin 79.

[DAV 75] B. DAVID.
*"Pour une généralisation des système C.A.O. Approche et Application".*
Thèse 3ème Cycle, GRENOBLE 75.

[DER 79] J.L. DERCHE (SLIGAS).
*"Outils de mise au point pour logiciels de micros".*
pp. 119-125.

[DEM 78] M. DEMUYCK et B. MEYER.
*"Les langages de spécification".*
PONT-A-MOUSSON, pp. 77-112, 20-22 Fév. 79 pp. 247-256.
Journées Génie Logiciel.

[EBO 76] M. EBOUEYA.
*"Moniteurs logiciels"*
D.E.A. 76, Lille 1.

[LAT 76] J.C. LATOMBE.
*"Artificial intelligence in CAD : The TROPIC System".*
Proceeding IFIP WC 52 ON CAD systems.
North-Holland Publishing compagny 1976.

[MAC 76] T.J. Mc CABE.
*"A complexity mesure".*
IEEE transaction on soltware engineering. Vol. SE 2 N° 4,
pp. 308-320, Déc. 76.

[MEM 79] G. MEMMI (ECA automator).
"Mesures qualitatives et quantitatives du logiciel".
Journées Génie Logiciel, pp. 247-256.

[OGD 78] C.A. OGDIN.
*"Software Desigh for µC's".*
Prentice Hall International Inc. London 78.

[RAV 79] Bruce W. RAVENEL.
*"Toward a Pascal Standard".*
Computer. April 1979.

[TEI 79] D. TEICHROEW et H. SAYANI.
*"Méthode ISDOS : Automatisation de la construction de systèmes".*
L'INFORMATIQUE, Déc. 71.

[WAT 76] S.J. WATTERS.
*MAO (CAM) Méthodologie Assistée par Ordinateur dans la conception des systèmes informatiques".*
L'INFORMATIQUE NOUVELLE, N° 69, Janvier 1976.

[ZAC 77] M. ZACHARIADES.
*"MAS : Réalisation d'un langage d'aide à la description et à la conception de systèmes logiques".*
Thèse, GRENOBLE, 1976.

# CONCLUSION

Les problèmes posés lors de la conception d'un SBμP peuvent se résumer à :

1° la difficulté d'obtenir de l'utilisateur un énnoncé clair de son application. C'est le problème de rédaction du cahier des charges. Quelque soit sa formulation, elle doit permettre d'aboutir au cahier des charges structuré. Cet aspect est celui abordé au cours des deux premiers chapitres.

2° La définition et la réalisation de procédures fonctionnelles susceptibles de permettre la mise en oeuvre du système utilisateur. Nous faisons une proposition de noyau au chapitre 3.

3° La nécessité d'une description progressive des spécifications fonctionnelles et technologiques, concernant le problème de l'utilisateur.

Nous avons étudié en détail ces sujets, en vue de permettre au concepteur :

1° une meilleure compréhension de ce qu'est un μθ ; ce qui permet de prévoir les modules nécessaires à l'application. En effet un μθ peut se décomposer en macro-modules dont l'assemblage permet d'effectuer des traitements souhaités.

2° la possibilité à chaque phase de conception, d'une vérification de conformité aux spécifications initiales. Vérification qui passe par des entretiens avec l'utilisateur, et où les modules d'estimation de temps et de taille trouvent leurs utilisations.
Au fur et à mesure que les logiciels de résolution des sous-problèmes de l'application sont construits, la compatibilité des solutions partielles vis à vis du problème initial est à vérifier.

3° Un choix rationnel pouvant être justifié techniquement, compte tenu du nombre grandissant de matériels compétitifs de SBµP.

4° Assemblage des modules matériels, vérification de la configuration logiciel/matériel dans l'environnement réel.

Nous n'avons pas abordé les problèmes de conception qui trouvent leurs solution dans les systèmes de développement existants. Nous ne nous sommes pas attardés aux conseils pratiques de réalisation, que le lecteur trouvera dans l'abondante littérature micro-informatique. Car souvent c'est soit de la publicité déguisée, soit un conseil propre à un cas très particulier.

COSMIC consitue sur le papier, une aide à la conception de systèmes à base de microprocesseur. Nous n'avons fait aucune mesure concernant ses performances. Il reste donc pour un travail ultérieur tout l'aspect réalisation pratique, c'est-à-dire, l'implémentation :

* d'un compilateur du langage de définition. Le compilateur devra produire les structures de données définies au chapitre 3

* de la partie conversationnelle du langage de définition

* du module chargement lequel crée les liens nécessaires entre la description de l'application, le noyau logiciel et les modules logiciels définis dans le catalogue.

* Des algorithmes de choix et d'estimations

* des primitives liés au noyau logiciel parmi lesquels nous trouvons :

    + les primitives d'échange et synchronisation
    + les primitives de génération d'interfaces avec les périphériques

* de la base des données : la structure, les noyaux d'accès, les primitives de mise à jour, etc. Cette base de données doit contenir des renseignements sur :

    + les boitiers LSI élémentaires, les cartes, etc
    + les modules logiciels mis à la disposition du concepteur

Pour aborder cette dernière, il faudrait définir l'interface entre COSMIC et PYTHIE. Ceci puisque ils utilisent, à un certain moment de la phase de conception, des données communes : les caractéristiques de l'équipement microprocesseur de base.

# ANNEXE A

# SYNTAXE-SEMANTIQUE

# DU

# LANGAGE DE DEFINITION

Avant de passer à la description de la syntaxe du langage de définition nous allons définir quelques règles utilisés très fréquemment

<identification de α>::= <identifier>

<liste de α> ::=
<identification de α> /
<identification de α>, <liste de α>

<chiffre entier> ::= <entier sans signe>

Un programme écrit en termes du langage de définition à la structure suivante, selon la Forme Normale de Backus (FNB) :

<programme>::=
SPEC. FONCTIONNELLES <spécifications fonctionnelles>
SPEC. TECHNOLOGIQUES <spécifications technologiques>

<spécifications fonctionnelles> ::=
PARTIE.OPERATIVE <partie operative> FIN.PARTIE.OPERATIVE
PARTIE.CONTROLE <partie contrôle> FIN.PARTIE.CONTROLE

<partie opérative> :: = <description de tâche> /
<description de tâche> <partie opérative>

<description de tâche> ::=

TACHE <identification de la tâche> ;
PRIORITE <chiffre entière> ;
<déclaration des variables de communication> ;
<déclaration des variables locales> ;
<partie traitement> ;
FIN <identification de la tâche> ;

<déclaration des variables de communication> ::=
GLOBAL <déclaration des variables globales>
CONNEXION <déclaration des variables de connexion>

<déclaration des variables globales> ::=
<liste de variables globales> : <type des variables> ; /
/ <liste de variables globales> : <type des variables> ;
<déclaration des variables globales>

<déclaration des variables de connexion> ::=
<liste de variables de connexion> : <type des variables> ;
/ <liste de variables de connexion> : <type des vairables> ;
<déclaration des variables de connexion>

<déclaration des variables locales> ::=
VAR <liste de variables> : <type des variables>

La règle syntaxique <type des variables> correspond à la règle syntaxique type de PASCAL. Ainsi le concepteur peut déclarer les variables de type simple : entier, réel, etc ou des types complexes tableau, ensemble, fichier, structure, chaine , etc.

<partie traitement> ::=
DEBUT <liste d'instructions> FIN

<instruction> ::= <instruction simple> /
<instruction structurée>

La syntaxe complète d'instruction est celle suivi dans le manuel PASCAL / IRIS 80 [MAU 78], auquel le lecteur est prié de se rapporter.

<partie contrôle> ::=
<déclaration des variables de synchronisation>
DEBUT <énoncé des règles de synchronisation> FIN

La déclaration des variables de synchronisation suit la même syntaxe que celle des variables locales et de communication. Il s'agit ici de déclarer toutes les variables utilisés dans la partie contrôle.

<déclaration des variables de synchronisation> ::=
<déclaration des variables de communication>
<déclaration des variables locales>

<énoncé des règles de synchronisation> ::=
R<chiffre entier> : <règle de synchronisation> /
/ R<chiffre entier> : <règle de synchronisation>
<énoncé des règles de synchronisation>

L'identification de chaque règle de synchronisation est un moyen, pour le concepteur, de pouvoir adresser facilement une règle de synchronisation. Cette identification n'intervient ni dans la gestion des échanges, ni dans la gestion de la synchronisation.

<règle de synchronisation> ::= <règles conditionnelles>/
<règles répétitives>/<règles combinées>

<règle conditionnelle> ::= <instruction répétitive>

<règle conditionnelle simple>/ <règle conditionnelle simple>
:

<règle conditionnelle simple> ::=
SI <expression> ALORS <action élémentaire> /
SI <expression> ALORS <action élémentaire>
SINON <action élémentaire>/
CAS <expression> DANS <sélecteur>

La règle <expression> suit la syntaxe définit en [MAU 78]. La règle <instruction répétitive> suit la syntaxe de POUR.

<sélecteur> ::= <constante> : <action élémentaire> /
<constante> : <action élémentaire> ; <sélecteur>

<règles répétitives> ::= <action élémentaire>
TOUTES LES <constantes> <unité de comptage> /
<action élémentaire> APRES <constante><unité de comptage>

<règles combinées> ::=
<liste d'expressions de répétition>
<liste d'expression conditionnelles>
ALORS <action élémentaire>

<expression de répétition> ::=
TOUTES LES <constante><unité de comptage>

<expression conditionnelle> ::= SI <expression>

Les règles combinées servent à exprimer des règles qui peuvent se déclencher sur l'apparition d'une condition périodique ou d'un événement.
EXM :

toutes les N MCS ou toutes les R exec. de T1
ou si B ou si non A alors exécuter TACHE 1

<unité de comptage> ::= HRS/MIN/SEC/MLS/MCS/
EXEC. De <identification de tâche>

Nous n'avons développé que l'échange des variables et la synchronisation des tâches. Pour cette raison nous décrirons seulement les actions élémentaires qui concernent les tâches [cf. 3.2.2.3].

<action élémentaire> ::=

DESACTIVER <identification de tâche>
ACTIVER <identification de tâche>
EXECUTER <identification de tâche>
CREER <identification de tâche>
ATTENDRE <identification de tâche>

L'ensemble des spécifications technologiques concerne essentiellement la description des périphériques.

<spécifications technologiques> ::=
<description des périphériques> /
<description des périphériques>
<spécifications technologiques>

<description des périphériques> ::=

PERIPHERIQUE <identification du périphérique>
VITESSE <chiffre><unité de transfert>
CAPACITE <chiffre><unité de stockage>
GESTIONNAIRE <identification de la tâche gestionnaire>
DEBUT <description des lignes du périphérique>
FIN <identification du périphérique>

<unité de transfert> ::= BAUD/B*S/L*S/C*S/BLCK*S
<unité de stockage> ::= CAR/BITS/MOTS/KMOTS/
KBITS

<description des lignes périphériques> ::=
SHAKEHAND <description des lignes shakehaud>
COMMANDE <description des lignes de commande>
DONNEES <description des lignes de données>

Puisque ces trois types de lignes sont décrites de la même façon nous allons en décrire le générique en l'appellant β. Ainsi β substitue l'un des concepts SHAKEHAND ou COMMANDE ou DONNEES.

<description des lignes β> ::=
ENTREE <description des lignes β vers le périphérique>
SORTIE <description des lignes β en provenance des périphériques>

Il faut remarquer que la description des lignes du périphérique est faite vis-à-vis du périphérique. C'est-à-dire que les lignes dites d'entrée sont celles qui vont vers le périphérique.

<description des lignes β vers le périphérique> ::=
<liste des lignes β en entrée> : <type>

<description des lgines β en provenance des périphériques> ::=
<liste des lignes β en sortie> : <type>

# BIBLIOGRAPHIE

[BOW 77] BOWLES, K.L.
*Microcomputer problem solving using pascal*
Springer Verlag, 1977

[MAU 78] MAURICE, PIERRE
*Pascal 80 / Manuel d'utilisation*
Rapport IRIA, Décembre 1978

# ANNEXE B

## PROBLÈME DE LA FABRIQUE DE LARD

Nous allons présenter dans cette annexe, les aspects complémentaires que nous n'avons pas traité tout le long de la thèse.

Tout doit se passer de telle manière que chaque opérateur puisse saisir toutes les informations sur 3 cochons en une minute. Le temps mort étant de 5 secondes au maximum. Ainsi il faut moins de 20 s par cochon.
P = 20 s x 4 (cochons) = 800

Des caractéristiques des E/S, nous déduisont que :

- pour saisir les 25 caractères, il faut V(1) = 2,5 seconde, puisque cela se fait à la vitesse de 10 C/s (maximum)

- l'enregistrement sur disque dure 0,15 x 2 = 0,3 s (pour l'écriture et la vérification) V(4) = 0,3 s

- et il faut cinq secondes pour imprimer une ligne. V(5) = 5 s

- l'en-tête est imprimée en 18 secondes.

Avec 40 cochons par terminal (page), le temps moyen nécessaire pour l'impression d'un enregistrement est de 5, 5 secondes y compris l'en-tête :

18 : 40 # 0,45 s en-tête
5,00 s ligne
# 5,5

Avec trois opérateurs travaillant à plein, nous avons 16,5 secondes (5,5 × 3) d'impression tous les cyles de 20 secondes.

Pour avoir un temps de réponse satisfaisant, il faut faire des recouvrements entre les tâches [CAP 78] de prise en compte de la donnée au niveau des terminaux.

Nous avons précédemment fait remarquer qu'il était possible de décomposer l'événement, dans cet exemple en deux ou trois tâches.

a) 1° enregistrement de la donnée sur l'un des 3 terminaux
2° écriture sur disquette
3° impression du contenu d'un tampon

b) 1° enregistrement de la donnée sur terminal
2° écriture sur disquette et impression

Le choix entre l'une des 2 décompositions peut être basé sur l'analyse du temps moyen nécessaire pour la prise en compte d'un évènement.

Pour la solution a), il faut en moyenne 5,5 s pour un évènement, alors qu'il en faut (5,5 + 0,3) pour la solution b).
Chacune des deux solutions donne une durée acceptable.

a) 5,5 × 3 = 16, 5 secondes

b) 5,8 × 3 = 17,4 secondes

car inférieure à 20 secondes, si le nombre n de registres tampons est suffisant. Les données transittent en effet par ces tampons, qui sont les boites à lettres entre le consommateur et les producteurs.

Les tâches feront appel au module d'échange (donc le noyau) à plusieurs moments de leur exécution. Il existe des moments ou le "passage de main" entre tâches est invisible, alors le recouvrement est possible. Par exemple une tâche sera mise en attente de l'imprimante (qui est active), pendant qu'une seconde tâche attend que la disquette soit prête, et que la troisième tâche utilise le µP pour traiter un caractère qui vient d'être lu d'un terminal.

La seule interférence est celle où toutes les tâches attendent à la section critique (accès à la disquette et à l'imprimante).

Dans le cas le plus défavorable, on a :

0,3 * 3 = 0,9 secondes d'attente lorsque 2 tâches utilisent la disquette et font un transfert

5 * 3 = 15 s pour que les caractères des trois terminaux soient imprimés

et 18 s s'il y a lieu de faire l'entête

soit au total 33,9 s, comme temps de réponse à un évènement, ce qui est beaucoup par rapport aux 5 secondes de temps de réponse imposé.

Un évènement étant ici composé de l'enregistrement de la donnée, son écriture sur disquette et son impression sur papier.

Même s'il n'y avait qu'un seul terminal, le cas le plus défavorable serait :

0,3 s + 5 s + 18 s = 23,3 secondes, au minimum, ce qui est encore inacceptable.

$c_1$) Estimation de n

Si la période P est inférieure à 23,3 secondes, il n'y a aucun tampon libérable donc n = 0.

Ajoutons à 23,3 s les 5,3 s on a : si 23,3 < P < 28,6 s alors n = 1
si 28,6 < P < 23,9 s alors n = 2

$c_2$) Estimation sur r

| | | | |
|---|---|---|---|
| si | P | 5 s | le nombre maximum de tampons nécessaires est r = 3 |
| 5 | P | 25 | le nombre maximum de tampons nécessaires est r = 6 |
| 25 | P | 45 | le nombre maximum de tampons nécessaires est r = 9 |

Les 20 s ajoutées correspondent au traitement d'un cochon par terminal.

En regroupant les résultats de $c_1$ et $c_2$, on a le résultats énoncé dans [CAM 78] :

| | | |
|---|---|---|
| | P < 5 | 0+f ≥ 3 ⇒ f ≥ 3 |
| 5 ≤ | P ≤ 23,3 | 0+f ≥ 6 ⇒ f ≥ 6 |
| 23,3 ≤ | P ≤ 25 | 1+f ≥ 6 ⇒ f ≥ 5 |
| 25 ≤ | P ≤ 28,6 | 1 ≥ 9 ⇒ f ≥ 8 |
| 28,6 ≤ | P < 33,9 | 2+f ≥ 9 ⇒ f ≥ 7 |
| 33,9 ≤ | P < 39,2 | 3+f ≥ 9 ⇒ f ≥ 6 |

Il faut donc un minimum de f ≥ 8 tampons.

Détermination du nombre de tampons nécessaire f, pour respecter les contraintes de temps [CAM 79].

## a) RAPPEL DES CONTRAINTES :

La durée entre :

la frappe du dernier caractère d'information sur un cochon i, et la frappe du premier caractère d'information pour le cochon suivant i+1, sur le même terminal, ne doit pas dépasser 5 s.

## b) CALCULS

Soient :

- n le nombre minimum de tampons libérables par la tâche commune
- r le nombre maximum de tampons nécessaires aux taches terminales
- f le nombre de tampons libres

Pour une période P de temps on doit avoir :

$$n + f \geq r$$

## A.2 SPÉCIFICATIONS FONCTIONNELLES

Nous donnons ci-après, la signification des variables utilisées lors de la programmation des tâches.

T-BOOL [I] témoigne du fait que le tampon numéro I a été rempli (BOOL [I] = vrai) par une des tâches TERMINAL.
Si BOOL [I] = faux, alors le tampon I n'a pas encore été rempli.

RESERVE [I] témoigne du fait que le tampon numéro I a été réservé (RESERVE [I] = vrai) par une des 3 tâches TERMINAL. Si RESERVE [I] = faux, alors le tampon I n'est pas réservé et la tâche TERMINAL qui le demande peut le réserver.

TAMPON est un tableau de variables du type type.tampon : ce type.tampon étant construit comme suit :

```
type.tampon = record
                  POIDS.COCHON : réel
                  SERIAL : entier
                IDENTIFICATION.COCHON : entier
                IDENTIFICATION.FOURNISSEUR : entier
                AGE.COCHON : entier
              end
```

NB-TAMPON est le nombre de tampon à utiliser.

CHOISI désigne le TAMPON plein, choisi pour être transféré.

choisir (NUMERO.DE.TAMPON) est une primitive appelée par les tâches DAC-1, DAC 2, DAC 3 en EXCLUSION MUTUELLE. Cette primitive retourne le numéro de tampon réservé à la tâche appelante. Elle est défini de la façon suivante :

```
procédure CHOISIR (NUMERO.DE.TAMPON : entier)
début
     P(mutex) ; pour k : = 1 jusqu'à NB-TAMPON
     faire
si  T-BOOL [k] = faux et RESERVE [k] = faux
alors début RESERVE [k] = vrai ;
            NUMERO.DE.TAMPON : k ;
            envoyer RESERVE [k] ;
      fin
fin faire
 fin
```

NUMERO DE TAMPON est le numéro du tampon réservé par la tâche appelante.

partie.opérative

```
tâche COMMUN
priorité :
          global NB-TAMPON : entier,
                 TAMPON : tableau [1..NB-TAMPON]
                          de type.tampon,

                 T BOOL : tableau [1..NB-TAMPON]
                          de.booleen ;
          connexion CHOISI : entier
traitement
     VAR ENTETE : chaine, CHARATER : char
début
(* transférer le tampon sur la disquette *)

     écrire TAMPON [CHOISI],
     tester.l'écriture,
     attendre.fin.d'écriture.sur.disquette,
```

```
(* convertir le tampon en format imprimante *)

      convertir TAMPON [CHOISI],

(* libérer le tampon et informer la tâche que choisi les tampons pleins *)
      T-BOOL [CHOISI] = faux,
      envoyer (T-BOOL [CHOISI]),
        :
(* imprimer le tampon *)
      si nécessaire imprimer EN_TETE,
      faire imprimer CHARATER
      jusqu'à ligne.complète

fin COMMUN

tâche DAC 1 (* manipule les données issues du convertisseur *)
      global NB-TAMPON : entier,
             T-BOOL : tableau [1..NB-TAMPON] booleen
             RESERVE : tableau [1..NB-TAMPON] booleen

traitement

      VAR VALEUR : tableau [1..NB-TAMPON] ; k : integer ; POIDS : réel ;
début

      (* lire la valeur rangée en mémoire par le convertisseur *)
                lire VALEUR
      (* convertir cette valeur dans un nombre réel *)
                POIDS = convertir (VALEUR)
      (* choisir un tampon vide à remplir *)
                choisir (NUMERO.DE.TAMPON)
      (* mémoriser le poids dans le tampon élu *)
                mémoriser (POIDS, NUMERO.DE.TAMPON)
      (* transmettre le numéro de tampon à la tâche TERMINAL associé *)
                envoyer (NUMERO.DE.TAMPON)

fin DAC
```

```
tâche DAC 2 (* manipule les données issues du convertisseur *)
      global NB-TAMPON : entier,
             T-BOOL : tableau [1..NB-TAMPON] booleen
             RESERVE : tableau [1..NB-TAMPON] booleen

traitement

      VAR VALEUR : tableau [1..NB-TAMPON] ; k : integer ; POIDS : réel ;
début

      (* lire la valeur rangé en mémoire par le convertisseur *)
                   lire VALEUR
      (* convertir cette valeur dans un nombre réel *)
                   POIDS = convertir (VALEUR)
      (* choisir un tampon vide à remplir *)
                   choisir (NUMERO.DE.TAMPON)
      (* mémoriser le poids dans le tampon élu *)
                   mémoriser (POIDS, NUMERO.DE.TAMPON)
      (* transmettre le numéro de tampon à la tâche TERMINAL associé *)
                   envoyer (NUMERO.DE.TAMPON)

fin DAC


tâche DAC 3 (* ménipule les données issues du convertisseur *)
      global NB-TAMPON : entier,
             T-BOOL : tableau [1..NB-TAMPON] booleen
             RESERVE : tableau [1..NB-TAMPON] booleen

traitement
      VAR VALEUR : tableau [1..NB-TAMPON] ; k : integer ; POIDS : réel
début
      (* lire la valeur rangée en mémoire par le convertisseur *)
                   lire VALEUR
      (* convertir cette valeur dans un nombre réel *)
                   POIDS = convertir (VALEUR)
```

```
        (* choisir un tampon vide à remplir *)
                choisir (NUMERO.DE.TAMPON)
        (* mémoriser le poids dans le tampon élu *)
                mémoriser (POIDS, NUMERO.DE.TAMPON)
        (* transmettre le numéro de tampon à la tâche TERMINAL associé *)
                envoyer (NUMERO.DE.TAMPON)

fin DAC :


tâche TERMINAL 1
        global NB-TAMPON : entier
                RESERVE : tableau [1..NB-TAMPON] de.booleen
                T-BOOL : tableau [1..NB-TAMPON] de.booleen
        connexion NUMERO.DE.TAMPON 1 : entier

traitement
        VAR CHARAT : character
début
        (* saisir les données du clavier et ranger le character dans le tam-
        pon approprié *)
                faire lire CHARAT
                        mémoriser (CHARAT, NUMERO.DE.TAMPON 1)
                jusqu'à dernier.character
        (* libérer le tampon rempli *)
                T-BOOL [NUMERO.DE.TAMPON 1] = vrai
                envoyer (T-BOOL [NUMERO.DE.TAMPON 1])

fin TERMINAL
```

```
tâche TERMINAL 2
      global NB-TAMPON : entier
             RESERVE : tableau [1..NB-TAMPON] de.booleen
             T-BOOL : tableau [1..NB-TAMPON] de.booleen
      connexion NUMERO.DE.TAMPON 2 : entier

traitement
      VAR CHARAT : character
début
      (* saisir les données du clavier et ranger le character dans le tam-
      pon approprié *)
            faire lire CHARAT
               mémoriser (CHARAT, NUMERO.DE.TAMPON 2)
            jusqu'à dernier.character
      (* libérer le tampon rempli *)
            T-BOOL [NUMERO.DE.TAMPON 2] = vrai
            envoyer (T-BOOL [NUMERO.DE.TAMPON 2])

fin TERMINAL


tâche TERMINAL 3
      global NB-TAMPON : entier
             RESERVE : tableau [1..NB-TAMPON] de.booleen
             T-BOOL : tableau [1..NB-TAMPON] de.booleen
      connexion NUMERO.DE.TAMPON 3 : entier

traitement
      VAR CHARAT : character
début
      (* saisir les données du clavier et ranger le character dans le tam-
      pon approprié *)
            faire lire CHARAT
                   mémoriser (CHARAT, NUMERO.DE.TAMPON 3)
```

```
            jusqu'à dernier.character
        (* libérer le tampon rempli *)
            T-BOOL [NUMERO.DE.TAMPON 3] = vrai
            envoyer  T-BOOL [NUMERO.DE.TAMPON 3])

fin TERMINAL


tâche SURVEILLANCE

        variables de communication et variables locales

début

    surveiller variables.d'état
    surveiller température
    surveiller niveau
    surveiller Pression

    si anomalie déclencher alarme

fin SURVEILLANCE


partie.contrôle
```

déclaration des variables de synchronisation

```
I ; NB-TAMPON ; T-BOOL ;
CHOISI ; START.CONVERSION ;
STROBE, de la même façon
que dans les tâches.
```

```
(* obtenir un buffer plein pour COMMUN *)

pour I = 1  JUSQU'A   NBTAMPON
début si TBOOL  I  = vrai
      alors CHOISI = I
      envoyer (CHOISI)
fin

(* démarrage des périphériques CONVERTISSEUR A/D *)

si START. CONVERSION 1 alors activer CONVERSION 1
si START. CONVERSION 2 alors activer CONVERSION 2
si START. CONVERSION 3 alors activer CONVERSION 3

(* démarrage des tâches de manipulation des données issues des convertisseur *)

si STROBE1 = vrai alors activer DAC1
si STROBE2 = vrai alors activer DAC2
si STROBE3 = vrai alors activer DAC3

(* la tâche de surveillance est déclenchée lorsqu'il n'y a pas de
   travail de pesage et saisie de données *)

si aucun.porc ou toutes. les 24 heures
activer SURVEILLANCE

PARTIE. CONTROLE
```

## A.3 SPECIFICATIONS TECHNOLOGIQUES

Le problème a été finalement décomposé en 4 tâches : une tâche appellée COMMUN qui se charge d'imprimer et d'écrire sur la disquette et trois tâches TERMINAL (TERMINAL 1, TERMINAL 2 et TERMINAL 3).

Il y a alors 8 périphériques : 3 balances, 3 claviers, 1 disquette et 1 imprimante. Nous allons décrire seulement la tâche COMMUN, une des tâches TERMINAL (les deux autres étant semblables) et un des périphériques : la balance qui n'est plus qu'un convertisseur analogique digital 12 bits.

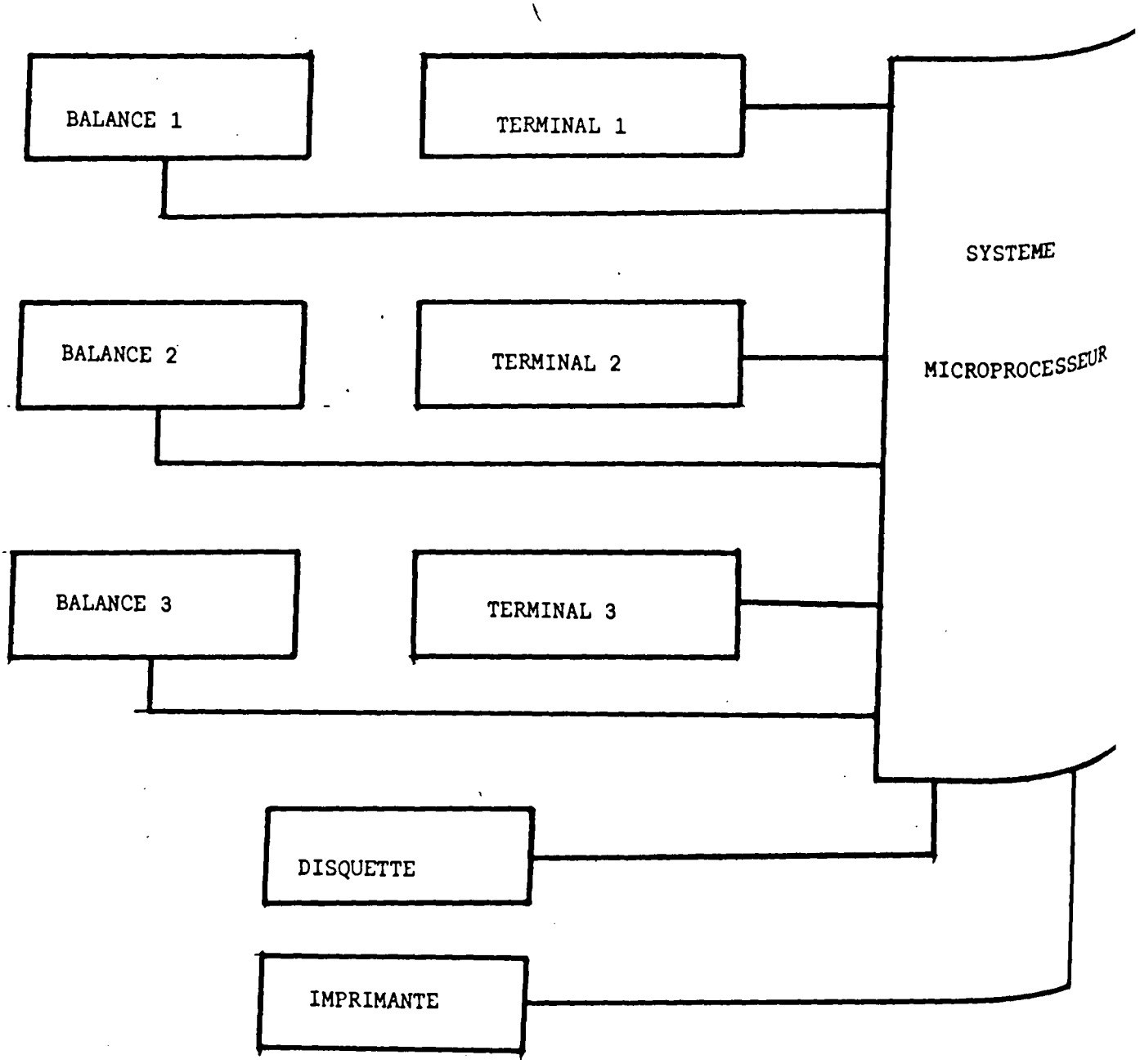
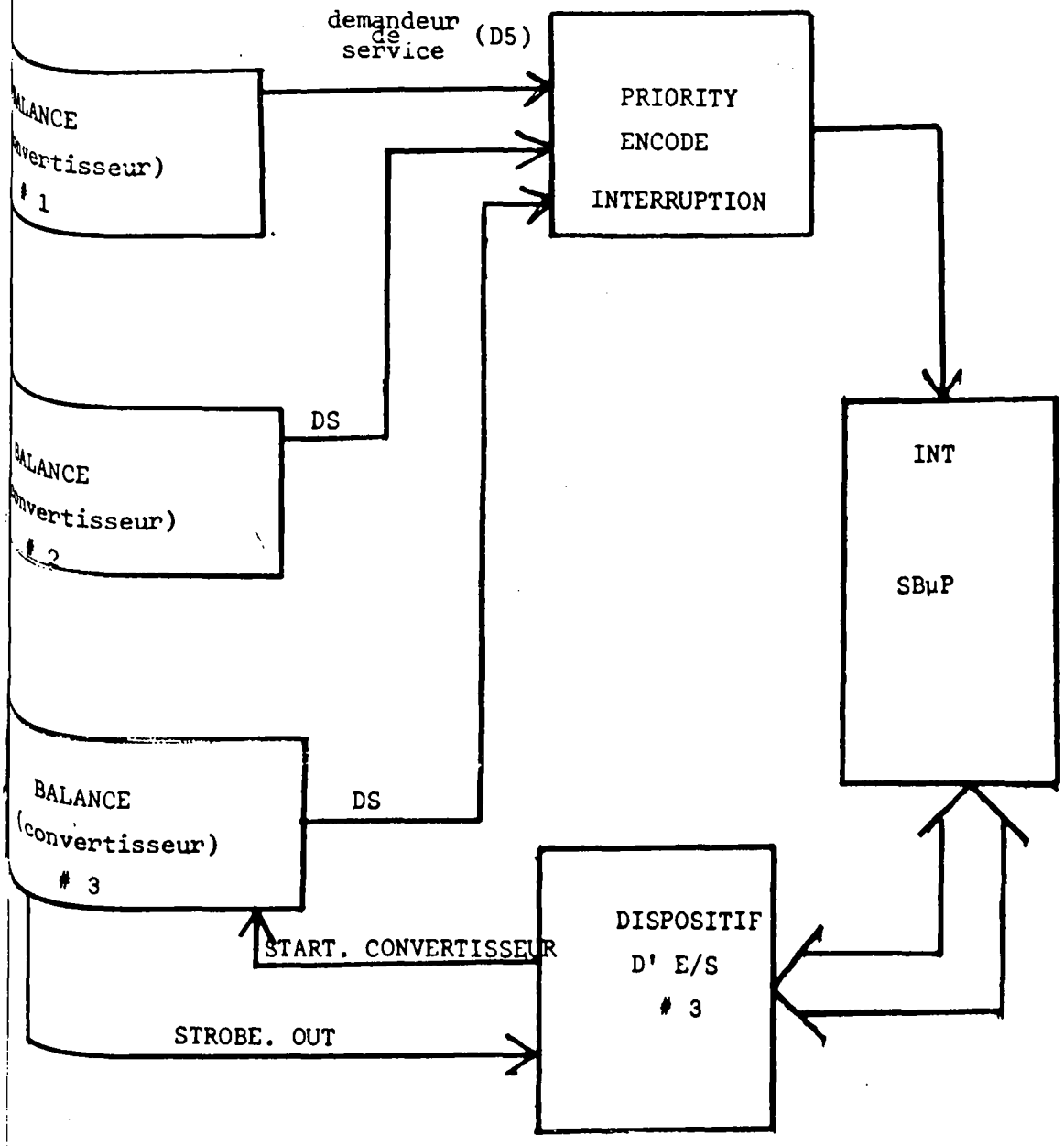



FIG. PERIPHERIQUES MIS EN OEUVRE DANS L'APPLICATION

Il nous aurait fallu décrire chacun des périphériques qui possède le concepteur (disquette, imprimante et terminaux) si ce ne sont pas des périphériques dont la description est déjà cataloguée.

Pour montrer comme l'utilisateur décrit ses périphériques le lecteur est invité à se rapporter au chapitre 4.

Le démarrage des périphériques convertisseur analogique/digitale 12 bits, nous conduit au schéma suivant :



Dès qu'un cochon est sur la balance une interruption est déclenchée. Le SPμP répond en envoyant, par l'intermédiaire du dispositif d'interface d'E/S, le signal START. CONVERSION. Lorsque la conversion est faite, le périphérique déclenche le signal STROBE.OUT 1 qui active la tâche DAC 1. Cette tâche récupère la valeur rangé en mémoire par le gestionnaire du périphérique CONVERTISSEUR 1.

---

# BIBLIOGRAPHIE

[CAP 78] CAPRINI O., LAUESEN S., OUGAARD U.
*"Design principles for dedicated data collection problems"*
EUROMICRO, 1978

[WIR 77] WIRTH NIKLAUS
*"Toward a discipline of real time programming"*
CACM, VOL 20,# 8, August 1977

# ANNEXE C

# LE POINT SUR LES MICROPROCESSEURS ET LEURS CARACTÉRISTIQUES

| Manufacturer | Processor | Process technology | Word size (data/instr.) | Direct addressing range | Number of basic instr. | Maximum clock frequency (MHz) | Instruction shortest/typical (µs) | TTL compatible | BCD arithmetic | On-chip interrupts/levels | Number of general purpose registers | Number of stack levels | On-chip clock | DMA capability | Specialized I/O circuits | Prototyping system available | Package size (pins) | Voltages required (V) | Assembler/ development | High level languages | Time share cross soft | Comments | Circle number |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Motorola | MC14500 | CMOS | 1/4 | 0 | 16 | 1/1 | 1/1 | Yes | No | Yes/1 | 1 | 0 | Yes | No | No[4] | No | 16 | 3 to 18 | No | No | No | Needs external program counter | 451 |
| Intel | 4004 | PMOS | 4/8 | 4k | 46 | 0.74/2 | 10.8/21.6 | No | Yes | Yes/1 | 16 | 3x12 | No | No | Yes | No | 16 | 15 | Yes | Yes | Yes | Superseded by 4040 | 452 |
| Intel | 4040 | PMOS | 4/8 | 8k | 60 | 0.74/2 | 10.8/21.6 | No | Yes | Yes/1 | 24 | 7x12 | No | No | Yes | Yes | 24 | 15 | Yes | Yes | Yes | General purpose 4-bit µP | 453 |
| NEC Microcomputers | µPD541 | PMOS | 4/8 | 4k | 69 | 0.5/2 | 6.4/38.4 | Yes | Yes | Yes/8 | 4 | 8x12 | No | Yes | Yes | Yes | 42 | 5,−5 | Yes | No | No | Intended for electronic cash registers, etc. | 454 |
| Fairchild | 2 chip F8 (3850) | NMOS | 8/8 | 64k | 69 | 2/1 | 2/13 | Yes | Yes | Yes/1 | 64 | RAM | Yes | Yes | Yes | Yes | 40 | 5,12 | Yes | Yes | Yes | Usually used with program storage unit | 455 |
| General Instrument | 8000 | PMOS | 8/8 | 1k | 48 | 0.8/2 | 1.25/3.75 | No | Yes | Yes/1 | 48 | 0 | No | No | Yes | Yes | 40 | 5,−12 | No | Yes | Yes | Predecessor of F8 | 457 |
| Intel | 8008 | PMOS | 8/8 | 16k | 48 | 0.8/2 | 12.5/37.5 | No | Yes | Yes/1 | 6 | 7x14 | No | No | Yes | Yes | 18 | 5,−9 | Yes | Yes | Yes | Predecessor of 8080, still in wide use | 458 |
| Intel | 8035/8039 | NMOS | 8/8 | 64k | 96 | 6/1 | 2.5/5 | Yes | Yes | Yes/1 | 64 | RAM | Yes | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | ROM-less versions of 8048/8049 | 448 |
| Intel | 8080A | NMOS | 8/8 | 64k | 78 | 2.6/2 | 1.5/3.75 | Yes[3] | Yes | Yes/1 | 8 | RAM | No | Yes | Yes | Yes | 40 | 5,12,−5 | Yes | Yes | Yes | By and large, still the most popular | 459 |
| Intel | 8085 | NMOS | 8/8 | 64k | 80 | 5.5/1 | 0.8/5.2 | Yes | Yes | Yes/4 | 8 | RAM | Yes | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | 8080 code compatible, has built-in clock | 460 |
| MOS Technology | MCS-650X | NMOS | 8/8 | 64k | 56 | 4/1 | 0.5/3.5 | Yes | Yes | Yes/1 | 0 | RAM | Yes | No | Yes | Yes | 40 | 5 | Yes | Yes | Yes | Provides 13 addressing modes | 461 |
| MOS Technology | MCS-651X | NMOS | 8/8 | 64k | 56 | 4/2 | 0.5/3.5 | Yes | Yes | Yes/1 | 0 | RAM | No | No | Yes | Yes | 40 | 5 | Yes | Yes | Yes | Similar to 650X but needs 2φ clock | 462 |
| Motorola | M6800 | NMOS | 8/8 | 64k | 72 | 2/2 | 1/2.5 | Yes | Yes | Yes/1 | 0 | RAM | No | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | Available in new depletion load version | 463 |
| Motorola | M6802 | NMOS | 8/8 | 64k | 72 | 2/1 | 2/5 | Yes | Yes | Yes/1 | 0 | RAM | Yes | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | Has 128 x 8 on chip RAM | 464 |
| Motorola | 6803/6808 | NMOS | 8/8 | 64k | 82 | 3.58/1 | 2/12 | Yes | Yes | Yes/1 | 128 | RAM | Yes | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | Has on-chip UART and counter/timer | 449 |
| Motorola | M6809 | NMOS | 8/8 | 64k | 59 | 2/1 | 2/5 | Yes | Yes | Yes/1 | 0 | RAM | Yes | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | Enhanced 6800 command set | 465 |
| National Semiconductor | INS8060/SC/MPH | NMOS | 8/8 | 4k | 46 | 4/1 | 5/10 | Yes | Yes | Yes/1 | 0 | RAM | Yes | Yes | No[4] | Yes | 40 | 5,−7 | Yes | Yes | Yes | Has handy daisy-chain capability | 466 |
| NEC Microcomputers | µPD 8080A | NMOS | 8/8 | 64k | 78 | 2/2 | 1.92/8.16 | Yes[3] | Yes | Yes/1 | 8 | RAM | No | Yes | Yes | Yes | 40 | 5,12,−5 | Yes | Yes | Yes | Pin compatible but does BCD subtraction | 467 |
| RCA | 1802 | CMOS | 8/8 | 64k | 91 | 6.4/1 | 2.5/3.75 | Yes | Yes | Yes/1 | 16 | RAM | Yes | Yes | Yes | Yes | 40 | 3 to 12 | Yes | Yes | Yes | Superseded two-chip version | 468 |
| RCA | 1803 | CMOS | 8/8 | 64k | 91 | 6.4/1 | 2.5/3.75 | Yes | Yes | Yes/1 | 16 | RAM | Yes | Yes | Yes | Yes | 28 | 3 to 12 | Yes | Yes | Yes | Trimmed down version of 1802 | 469 |
| Scientific Microsystems | SMS-300 | Bi-polar | 8/8 | 8k+ | 8 | 10/1 | | Yes | No | No | | 0 | No | | Yes | | 50 | | No | Yes | Yes | Very specialized instruction set | 470 |
| Signetics | 2650 | NMOS | 8/8 | 32k | 75 | 2/1 | 1.5/6 | Yes | Yes | Yes/1 | 7 | 8x15 | No | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | There are 1.25 and 2 MHz versions | 471 |
| Signetics | 8X300 | Bi-polar | 8/16 | 8k | ? | 4/1 | 0.25 | Yes | No | No | 8 | 0 | No | No | No[4] | Yes | 50 | 5 | Yes | No | Yes | Intended for high speed controllers | 450 |
| Zilog | Z80 | NMOS | 8/8 | 64k | 150+ | 4/1 | 1/5.75 | Yes | Yes | Yes/1 | 14 | RAM | No | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | 8080 instructions are a subset | 472 |
| Intersil | 6100 | CMOS | 12/12 | 4k | 81 | 4/1 | 2.5/5.5 | Yes | No | Yes/1 | 0 | RAM | Yes | Yes | Yes | Yes | 40 | 4 to 11 | Yes | Yes | Yes | Emulates PDP-8 instruction set | 473 |
| Toshiba | T3190 | PMOS NMOS | 12/12 | 4k | 108 | 2.5/1 | 10/30 | Yes | No | Yes/8 | 8 | RAM | Yes | Yes | Yes | Yes | 36 | 5,−5 | Yes | Yes | Yes | Has multiply and divide inst. | 474 |
| Data General | mN601 | NMOS | 16/16 | 32k | 42 | 8.33/2 | 1.2/29.5 | Yes | No | Yes/1 | 4 | RAM | Yes | Yes | Yes | No | 40 | 5,10,14,−4.25 | Yes | Yes | Yes | Emulates NOVA instruction set | 475 |
| Fairchild | 9440 | I²L | 16/16 | 64k | 42 | 10/1 | | Yes | No | Yes/1 | 4 | RAM | Yes | Yes | No[4] | No | 40 | | No | No | No | Emulates NOVA instruction set | 476 |
| Ferranti | F100L | Bi-polar | 16/16 | 32k | 153 | 14/1 | 1.19/14 | Yes | No | Yes/1 | 0 | RAM | No | Yes | Yes | Yes | 40 | 5,1.2 | Yes | Yes | Yes | Can do double word operations | 456 |
| General Instrument | CP1600/1610 | NMOS | 16/16 | 64k | 87 | 4/2 | 1.6/4.8 | Yes | No | Yes/1 | 8 | RAM | No | Yes | Yes | Yes | 40 | 5,12,−3 | Yes | Yes | Yes | All internal registers can be accumulators | 477 |
| Intel | 8086 | NMOS | 16/16 | 1M[5] | 97 | 5/1 | 0.4/37.8 | Yes | Yes | Yes/1 | 8 | RAM | Yes | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | Has 24 addressing modes | 541 |
| Motorola | MC68000 | NMOS | 16/16 | 16M[5] | 61 | 8/1 | 0.5/? | Yes | Yes | Yes/1 | 16 | RAM | No | Yes | Yes | Yes | 64 | 5 | Yes | Yes | Yes | Has 32-bit wide internal structure | 542 |
| National Semiconductor | INS8900 | NMOS | 16/16 | 64k | 45 | 2/2 | 2.5/5 | No | Yes | Yes/6 | 4 | 10x16 | No | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | Architecture intended for data handling | 478 |
| Panafacom | MN1610 | NMOS | 16/16 | 64k | 33 | 2/2 | 2/6 | Yes[3] | No | Yes/3 | 5 | RAM | No | Yes | Yes | No | 40 | 5,12,−3 | Yes | No | No | | 479 |
| Texas Instruments | TMS9930/9981 | NMOS | 16/16[1] | 8k | 69 | 4/4 | 3.2/49.6 | Yes[3] | No | Yes/4 | 16 | RAM | Yes | Yes | Yes | No | 40 | 5,12,−5 | Yes | Yes | Yes | The 9981 requires external clock | 480 |
| Texas Instruments | TMS9985 | NMOS | 16/16[1] | 32k | 68 | 5/1 | 2.4/50 | Yes | No | Yes/4 | 0 | RAM | Yes | Yes | Yes | Yes | 40 | 5 | Yes | Yes | Yes | ROM-less version of 9940, with buses | 543 |
| Texas Instruments | TMS/SBP9900 | NMOS/ I²L | 16/16 | 32k | 69 | 4/4 | 2/31 | Yes[3] | No | Yes/16 | 16 | RAM | No | Yes | Yes | No | 64 | 5,12,−5* | Yes | Yes | Yes | Emulates 990 mini instructions | 481 |
| Western Digital | WD-16 | NMOS | 16/16 | 64k | 116 | 3.3/4 | 2.1/780 | Yes | Yes | Yes/16 | 6 | RAM | No | Yes | Yes | Yes | 40 | 5,12,−5 | Yes | Yes | No | Very similar to DEC LSI-11 | 529 |
| Zilog | Z8000 | NMOS | 16/16 | 48M[5] | 110+ | 8/1 | 0.75/90 | Yes | Yes | Yes/1 | 16 | RAM | No | Yes | Yes | Yes | 40/48 | 5 | Yes | Yes | Yes | Two versions available | 544 |

1. Has 8-bit external buses and 16-bit internal buses. 2. With maximum clock. 3. Except clock lines. 4. Standard TTL or MOS circuits will suffice. 5. Range in bytes. *9980 only.

Et on trouve sur le marché les cartes résumées ci-dessous.

| Manufacturer | Model | Word size in bits (data/address) | CPU type | Clock freq. (MHz) Min./Max. | Memory bytes (k=1024): Total addressable | Amount of RAM on card (b=bytes) | Amount of ROM on card* | DMA capability | Bus type (P=proprietary, blank=no bus) | Parallel I/O lines | Serial I/O: Number of ports | Serial I/O: Baud rate (max.) (b baud) | Interrupt provisions | Multiprocessing capability | Counter-timers: No. of timers/bits per timer | Software: A Assembler; AP Applications Package; DB Debugging aids, monitor, etc.; H High level language(s); OS Operating system | Supply voltages (U=unregulated inputs; on-card regulators) | Board size (in.) | Comments | Circle number |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Advanced Micro Computer | SXC 85 | 8/16 | 8085A | 3 | 64 | 1.28 | 0/4 | | P | 44 | 1 | | • | | 2/14 | DB | +5 | 3.9×6.3 | 256 bytes CMOS RAM with battery backup. Eurocard version, SMP-80. | 431 |
| American Microsystems | EVK 300 | 8/16 | 6800 | 1 | 16 | 1 | 6 | • | | 56 | 1 | 9.6 | | | | A, DB, H, OS | +5 | 10.5×12 | Maximum addressable ROM 6k. EPROM programming for S6834. Disassembler available. EVK-100, EVK-200 also available. | 432 |
| Analog Precision | TPM2 | 8/16 | 8080A | 1.1/2 | 32 | 0.25 | 1.2 | | P | | | | | | | N/A | ±5, ±12 | 6×8 | 3-bit a/d converter. 1/0 port enables only. Power-on reset. | 433 |
| Apple Computer | A2B0004X | 8/16 | 6502 | 1.023 | 64 | 4/48 | 0/12 | | P | | 1 | 9.6 | | | | A, DB, H, AP, OS | ±5, ±12 | 8.5×14 | TTL serial interface, composite video output, cassette 1/0 port, ASCII keyboard input, audio output for loudspeaker, two joystick inputs. Parallel 1/0 is by bus accessory cards. | 434 |
| Applied Systems | ASC/80 | 8/16 | 8085 | 0.5/5 | 64 | 0.25/1 | 4 | | P | | 1 | | | • | | H | +5 | 4.5×6 | Software includes executive and emulation programs in PROMs; also communications software. Optional Z-80 CPU, multiprocessing capability, board size. | 435 |
| Compas Microsystems | CSB-1,2 | 8/16 | 6502 | 1/2 | 64 | 2 | 0/8 | | System 65 | 50/40 | 0/1 | 9.6 | | | | A, H, DB | +5 | 6×9.75 | Two boards differ only in I/O capabilities. | 436 |
| Computer Automation | LSI4/10 | 16/16 | Custom | 16 | 128 | 1/4 bwds | | • | P | 32 | | 19.2 | • | • | | A, DB, H, OS | +5 | 7.5×16.9 | Power fail shutdown and auto restart. Real-time clock. Pascal available. Multiply and divide instructions standard. | 482 |
| Control Logic | CSS-1143 | 8/16 | Z-80 | 2 | 64 | 1 | 0/16 | | P | | 1 | 110b | | | | A, AP, DB, H, OS | ±5, ±12 | 16×8 | User's program library available for both types. | 483 |
| | MM1-MSC | 8/16 | Z-80 | 2/4 | 64 | 1024b | 4 | | | | 4 | 50 | • | • | | A, AP, DB, H, OS | ±5, +12 | 10×7 | Serial ports TTL, to 9.6 kbaud asynch., 50 kbaud synch. Three interrupt inputs. | 484 |
| Creative Micro Systems | CMS 9600 | 8/16 | 6802 | 0.5/1 | 56 | 1.2 | 6.1 | | EXOR ciser | 40 | 2 | 38.4 | • | | 3/16 | DB, AP | ±5, ±12 | 6×9.75 | Compatible with Motorola EXORciser boards. | 485 |
| Cromemco | SCC-W | 8/16 | Z-80A | 4 | 8.512 | 1 | 0/8 | • | S-100 | 24 | 1 | 76.8 | | | 5/16 | A, DB, H, OS | +8 ±18 U | 5×10 | Mates with all S-100 compatible computers. | 486 |
| Data General | MBC/1 | 16/15 | mN601 | 8.3 | 32 | 2 | 0/4 | | P | 32 | 1 | 9.6 | | | | A, AP, SB, H, OS | +5, +15, -5 | 10.4×7.5 | Software includes most Nova and Eclipse programs. | 829 |
| | MicroNova | 16/15 | mN601 | 8.3 | 32 | 2/4 | 0.5/4 | | P | | 1 | 16.6M | | | | A, AP, SB, H, OS | +5, +15, -5 | 9.5×7.5 | I/O is bit serial at 16.6 MHz. Software includes most Nova and Eclipse programs. | 487 |
| Digital Eqpt. Corp. | KD11-F | 16/15 | LSI-11 | 2.6 | 32 | 4/4 | 0 | • | P | | 1 | | | | | A, AP, DB, H, OS | +5, +12 | 10.5×8.9 | Jumper-selected restart mode. Double-precision fixed and floating-point arithmetic and multiply/divide options. Software-compatible with PDP-11. | 488 |
| Digitek | DK-6800 | 8/16 | 6800 | 0.1/1 | 48 | 1 | 1/4 | | P | 16 | 1 | 19.2 | | | | OS, H, A, DB, AP | +5, +12 | 8×8.65 | Many additional support cards are available. | 489 |
| Dynabyte | BC1-1 | 8/16 | Z-80 | 2.5 | 64 | 8/16 | 0/12 | • | P | 88 | 4 | 9.6 | • | | | A, H | ±5, ±12, +28 | 14.8×12.4 | For control applications. 8 relay outputs, 16 LEDS, on-board EPROM programmer. Cassette port and video output. | 490 |
| Fairchild | OCM-1 | 8/16 | 3850 | 2 | 64 | 1 | 1/10 | | P | 64 | 1 | 110b | • | | 2/16 | DB | +5 | 7.5×10.5 | Includes TTY operating system and programmable baud rates. | 492 |
| | GLOW-16 | 16/16 | 9440 | 8/12 | 512 | 64 | 8 | • | Nova | 32 | 2 | 9.6 | • | | | A, H, DB | +5, +15 | 15×15 | Compatible with the Data General Nova series, and includes 16 switch-selectable baud rates. | 491 |
| | SPARK-16 | 16/16 | 9440 | 8/12 | 64 | 32 | 4/4 | • | Nova | 8 | 1 | 9.6 | • | | | DB | +5 | 8×10 | Smaller version of GLOW-16. | 493 |
| Honize Interactive Control | M-5002 | 8/16 | M6802 | 0.9216/2 | see comments | | 1/4 | • | P | 20 | 1 | 9.6 | • | • | | A, AP, OS | +5, ±12 | 9.75×5.98 | Maximum addressable memory 61,440 bytes RAM on card. 128 bytes provided. 1152 maximum. Power fail detection and restart. Optional battery backup for RAM. Accessory boards are Motorola-compatible. | 494 |
| Hourikon | MLZ-80 | 8/16 | Z-80 | 2 or 4 | 64 | 4 | 0/8 | • | Multibus | 32 | 2 | 19.2 | • | • | 4/16 | A, HL, OS | +5, ±12 | 6.75×12 | On-card floppy-disc controller. Power-on jump to selectable start address. Bus watchdog timer. 16 Mbyte address capability possible. SDLC compatible, with hardware CRC. System bus not required for on-card operations. | 496 |
| Hewlett-Packard | 2108K | 16/16 | Custom | 18.5 | 2M | see comments | | • | 21MX | 16 | 0 | | • | | • | A, AP, DB, H, OS | +5 -2 | 18.13×13 | Maximum addressable RAM 1 M-word. User-microcodable 4 or 16 k-words of ROM on card is for microcode. ROM words are 24 bits. Can address 16 k-words of microcode off-card. DMA transfer to 2 M-wds/s. 16 words of RAM on card. Member of 21MX family. | 497 |
| | 2109K | | | 28.5 | | | | | | | | | | | | | | | | |
| Imsai | MPU-B | 8/16 | 8085 | 3 | 64/1M | 0.25 | 2/4 | • | S-100 | 8 | 2 | 56 | • | • | 3/16 | A, DB, H, OS | +8 ±18 U | 5.25×10 | One Mbyte of RAM addressable thru auxiliary controller. Power-on jump. | 498 |

*provided/max. possible

N/A: not available

| Manufacturer | Model | Word size in bits (data/address) | CPU type | Clock freq. (MHz) Min./Max. | Memory kbytes (k=1024) Total addressable | Amount of RAM on card (k-bytes) | Amount of ROM on card[a] | DMA capability | Bus type (P=proprietary, blank=no bus) | Parallel I/O lines | Serial I/O Number of ports | Serial I/O Baud rate (max.) (k baud) | Interrupt provisions | Multiprocessing capability | Counter-timers: No. of timers/bits per timer | Software A Assembler; AP Applications Package; DB Debugging aids, monitor, etc.; H High level language(s); OS Operating system | Supply voltages (U=unregulated inputs; on-card regulators) | Board size (in.) | Comments | Circle number |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intel** | iSBC 80/04 | 8/16 | 8085 A | 1.955 | see comments | 0.25 | 2/4 | | none | 22 | 1 | 4.8 | • | | 1/14 | A, DB, H, OS | +5 | 6.75×7.85 | Total addressable RAM 256 bytes; total addressable ROM 4kbytes. For stand-alone applications. Serial I/O via CPU's SID, SOD lines. | 499 |
| | iSBC 80/05 | 8/16 | 8085 A | 1.966 | 64 | 0.5 | 2/4 | • | Multibus | 22 | 1 | 4.8 | • | • | 1/14 | A, DB, H, OS | +5 | 6.75×12 | Same features as 04 but plugs into Multibus. | 500 |
| | iSBC 80/10A | 8/16 | 8080 A | 2.048 | 64 | 1 | 0/8 | • | Multibus | 48 | 1 | 38.4 | • | • | | A, DB, H, OS | ±5, ±12 | 6.75×12 | Second-sourced by National Semiconductor and other companies. | 501 |
| | iSBC 80/20 | 8/16 | 8080 A | 2.15 | 64 | 2 | 0/8 | • | Multibus | 48 | 1 | 38.4 | • | • | 2/16 | A, DB, H, OS | ±5, ±12 | 6.75×12 | iSBC 80/20-4: 4 kbytes of RAM on card. Up to 16 CPU's on bus. | 502 |
| | iSBC 80/30 | 8/16 | 8085 A | 2.76 | 64 | 16/16[1] | 0/9[1] | • | Multibus | 24/48[1] | 1 | 38.4 | • | • | 2/16 | A, AP, DB, H, OS | ±5, ±12 | 6.75×12 | Board has dual-port memory and socket for an 8041/8741 universal peripheral interface. | 503 |
| | iSBC 86/12 | 16/16 | 8086 | 5 | 1000 | 32 | 0/16 | • | Multibus | 24 | 1 | 38.4 | • | • | 2/16 | A, AP, DB, H, OS | ±5, ±12 | 6.75×12 | Has dual-port memory, 9 levels of vectored interrupt and a 1-Mbyte addressing range. | 504 |
| **MicroDaSys** | MD-690a | 8/16 | MC 6802 | 1 | 56 | 1152b | 1/10 | | S-100 | 16 | 1 | 24 | | • | | DB | +8, ±16 U | 5.38×10 | MC6809 CPU also available. MIKBUG-compatible monitor designed to interface with most fast memory-mapped video and graphics boards. | 505 |
| **Monolithic Systems** | MSC-8001 | 8/16 | Z-80 A | 4 | 64 | 4/8 | 0/16 | • | Multibus | 48 | 1 | | • | • | 2/16 | DB,OS | +5 | 6.75×12 | Z-80 based, Multibus- compatible microcomputer. | 507 |
| | MSC-8004 | 8/16 | Z-80 A | 2/4 | 64 | 4/32 | 0/16 | | Multibus | 48 | 1 | 56 | | • | 3/16 | A, DB, OS | ±5, ±12 | 6.75×12 | Board has math chip for floating point processing, one time does baud-rate generation for serial I/O. | 508 |
| **Mostek** | OEM-80 | 8/16 | MK 3880 | 0.005/2.5 | 64 | 4/64 | 0/25 | • | P | 40 | 1 | 9.6 | | • | 4/16 | A, DB, H, OS | +5,±12 | 12×8.5 | Same as Z-80 CPU, ROM and RAM address mapping. OEM-80: No ROM supplied. Available as a complete ROM-based prototype package. European card. 233×250 mm. | 509 |
| | MD-SBC-1 | 8/16 | Z-80 A | 0.5/4 | 64 | 0.2/2 | 0/8 | | STD | 40 | 0 | | | | see comments | A, DB | +5 | 4.5×6.5 | Counter/timers in lieu of I/O on some boards. | 510 |
| **Motorola** | M68MM01 | 8/16 | MC 6800 | 1 | 41 | 1 | 0/4 | • | EXORciser | 60 | | | | • | • | A, DB, H | +5, -12 | 9.75×5.98 | Has 3 PIAs, 120 I/O lines total. Suitable for control applications. Power-on reset. EXbug can be used. | 511 |
| | M68MM01A | 8/16 | MC 6800 | 1 | 64 | 1 | 0/8 | • | EXORciser | 40 | 1 | 9.6 | • | • | | A, DB, H | +5, ±12 | 9.75×5.98 | Has 2 PIAs. Power-on reset. EXbug can be used. | 512 |
| | M68MM01B | 8/16 | MC 6802 | 1 | 42 | 128b | 0/8 | • | EXORciser | 26 | 1 | 9.6 | • | • | 3/16 | A, DB, H | +5 | 9.75×5.98 | Battery can back up lower 32 bytes of RAM in low-power mode. Power-on reset. EXbug can be used. | 513 |
| | M68MM01B1A | 8/16 | MC 6802 | 1 | 64 | 128b 384b | 0/8 | • | EXORciser | 26 | 1 | 9.6 | • | • | 2/16 | A, DB, H | +5, ±12 | 9.75×5.98 | Battery can back up lower 32 bytes of RAM in low-power mode. Power-on reset. EXbug can be used. Built-in dynamic RAM refresh circuits. Power-on reset. EXbug can be used. | 514 |
| | M68MM01D | 8/16 | 6800 | 1 | 56 | 0/10 | 0/10 | | EXORciser | 8 | 1 | 28.8 | | | | A, AP, DB, H, OS | +5, ±12 | 6.15×9.75 | Software support through EXORciser development library. | 515 |
| | M68MM19 | 8/16 | 6809 | 2 | 51 | 2/8 | 0/12 | | EXORciser | 16 | 1 | 28.8 | | | | DB, H | +5, ±12 | 6.15×9.75 | RAM includes 2 kbytes with battery backup. | 516 |
| **Mupro** | MBC-80/10X | 8/16 | 8080 A | 2 | 64 | 1/15 | 0/23 | • | Multibus | 48 | 1 | 38.4 | • | | | DB | ±5, ±12 | 6.75×12+ | User software libraries and time-shared cross-software available. | 517 |
| | MBC-80/20X | 8/16 | 8085 A | 2.15 | 64 | 1/15 | 0/23 | • | Multibus | 48 | 1 | 38.4 | • | • | | A, DB, OS | ±5, ±12 | 6.75×12 | Extremely large PROM space for applications programs. | 518 |
| | MBC-80/40X | 8/16 | 8080 A | 2.15 | 64 | 1/15 | 0/23 | • | Multibus | 48 | 1 | 38.4 | • | • | 3/16 | A, DB, OS | ±5, ±12 | 6.75×12 | Hardware arithmetic processor, calendar, and peripheral processor included. | 519 |
| **National Semiconductor** | IMP-16C/L | 16/16 | NMP-16 | 5.7143 | 64 | 1 | 0/1 | • | P | 16 | 1 | | • | • | | OS | +5, -12 | 11×8.5 | Multiprocessing, and 4 DMA ports (to 1 kbyte) on -16L only. Battery backup for. External clock possible. ROMs extend instruction set. Hi-speed (97 kwds/s) block transfer instructions. | 520 |
| | ISP-8C/10 JN | 8/16 | NSM-8A/600 | 4 | 64 | 0.25 | 2/0.5 | | P | 5 | 1 | 9.6 | • | • | | DB, HL | +5 | 4.38×4.86 | SC/MP II. Delay instruction (132 ms maximum). Suffix NE. Eurocard. | 521 |
| | BLC-80/05 | 8/16 | 8085 A | 1.966 | 64 | 512b | 2/8 | | Multibus | 22 | 1 | CPU limited | • | | 1/16 | A, AP, DB, H, OS | +5 | 6.75×12 | Equivalent to Intel's iSBC-80/05 CPU board. | 522 |
| | BLC-80/07 | 8/16 | 8080 A | 2.048 | 64 | 512b | 0/4 | | Multibus | 24 | | | • | | | A, AP, DB, H, OS | ±5, ±12 | 6.75×12 | Reduced memory and I/O version of the BLC-80/10 board. | 523 |
| | BLC-80/10, 11, 12, 14 | 8/16 | 8080A | 2.048 | 64 | see comments | • | | Multibus | 48 | 1 | 38.4 | • | • | | DB | ±5, ±12 | 6.75×12 | BLC-80/10 is Intel iSBC-80/10 equivalent. BLC-80/11 is Intel -10A equivalent. BLC-80/14 is same as BLC-80/12, except has 4 k of RAM. | 524 |

| Manufacturer | Model | Word size in bits (data/address) | CPU type | Clock freq. (MHz) Min./Max | Memory kbytes (k=1024) Total addressable | Amount of RAM on card (b=bytes) | Amount of ROM on card* | DMA capability | Bus type (P=proprietary, blank=no bus) | Serial I/O Parallel I/O lines | Serial I/O Number of ports | Baud rate (max.) (b/baud) | Interrupt provisions | Multiprocessing capability | Counter-timers, No. of timers/bits per timer | Software A Assembler; AP Applications Package; DB Debugging aids, monitor, etc; H High level language(s); OS Operating system | Supply voltages (U=unregulated inputs, on card regulators) | Dimensions (in.) | Comments | Circle number |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Omnibyte | OB 8001 | 8/16 | 6800 | 1 | 64 | 1152b | 0/4 | | P | | 1 | 19.2 | | | | A, DB, H, OS | ±5, ±12 | 4.5×6.5 | Parallel I/O: 16 programmable lines. | 527 |
| Pragmatic Designs | CPU-I | 8/16 | 8085 A | 3 | 48 | 0.5 | 0/4 | | P | 22 | 2 | CPU limited | • | | 1/14 | | +5 | 6.5×4 | Serial port built from 8085 SID/SOD lines. Breadboard area. | 571 |
| Pro-log | PLS-800 | 8/16 | | 1/3 | 8 | 1/2 | 0/8 | | P | 40 | 1 | 9.6 | | | | A, AP, DB, H | ±5, +12 | 4.5×5.5 | Five models available: — two with 8080A CPU, one with 8085, one with 6800 and one with Z-80 CPU. Serial I/O available only on 8085-based card. | 800 |
| | 7801, 2, 3 | 8/16 | 8085<br>6800<br>Z-80 | 1/3.125 | 64 | 1/4 | 8/8 | • | STD | 0 | 1 | CPU limited | • | • | | | +5 | 6.5×4.5 | Three CPUs are functionally compatible; baud rate depends on µP. | 801 |
| Process Computer Systems | PCS 1806 | 8/16 | 8080A | 2 | 64 | 1 | 0/7 | • | Flexibus II | 16 | 1 | 9.6 | • | | 5/8 | A, DB, H, OS | ±5, ±12 | 10.5×8.5 | Power fail interrupt. | 802 |
| | PCS 1810 | 8/16 | 8080A | 2 | 64 | 1 | 0/3 | • | Flexibus II | 32 | 1 | 9.6 | • | | 5/8 | A, DB, H, OS | ±5, +5 to +30 | 10.5×8.5 | Power fail interrupt. | 803 |
| | PCS 1880 • | 8/16 | Z-80 | 4 | 64 | 1 | 0/6 | • | Flexibus II | 8 | 1 | 9.6 | • | | 1/16 | A, DB, H, OS | ±5, ±12 | 10.5×8.5 | Optional AMD 9511 hardware math chip with fixed, float, and conversions; logs, trig, etc. | 804 |
| Processor Technology | Sol-PC | 8/16 | 8080 A | 2.05 | 64 | 2 | 2 | • | S-100 | 16 | 1 | 9.6 | • | | | A, DB, H, OS | ±5, ±12 | 16×18 | Six configurations. All have 1024-character video out. 2K OS in ROM, audio cassette interface, keyboard interface. | 805 |
| Quay | 90/94 MPS | 8/16 | Z-80A | 2.5/4 | 64 | 5/65 | 1/14 | • | P | 64 | 1 | 9.6 | | • | 4/16 | DB, H | ±5, +12, +28 | 16.18×7.88 | CPU Z-80 or Z-30A. PROM programmer. Single-step circuits. Hardware breakpoints. Total board memory to 72 k. Backplane-independent. | 806 |
| | 90F/MPS | 8/16 | Z-80A | 2.5/4 | 79 | 64 | 1/14 | • | P | 36 | 1 | 9.6 | • | | 1 | H, DB, OS | ±5, +12, +28 | 16.175×7.875 | Paging on upper 16 kbyte of memory provides a total of 79 kbytes. | 808 |
| RCI/Dain | CMC 68/04 | 8/16 | 6802 | 1 | 64 | 384b | 0/4 | | P | 16 | 1 | | | | 1 | A, AP | +5 | 7.6×6.3 | Firmware package available for use with HP5004A signature analyzer. | 809 |
| | CMC 68/15 | 8/16 | 6800 | 1/2 | 64 | 2 | 0/8 | • | P | 32 | 1 | | • | | | A, DB | +5 | 7.6×11 | Software can be developed with the Motorola EXORciser. | 810 |
| Relational Memory | RSBC-280/24 | 8/16 | Z-80A | 3.9 | 1048 | 20 | 2/24 | • | Multibus | 16 | 2 | 550 | • | • | | A, DB | ±5, ±12 | 6.75×12 | Up to 22 kbytes of ROM can be replaced by static RAM; in-circuit emulator available. | 811 |
| Space Byte | 8085 CPU | 8/16 | 8085 | 3 | 64 | 0.25 | 0/6 | • | S-100 | 32 | 2 | 38.4 | • | • | 1/14 | A, AP, DB, H, OS | +8, ±16 U | 5×10 | Parallel I/O port is embedded for floppy disc interface. | 812 |
| Synertek | CP 118 | 8/16 | 6502 | 1 | 64 | 1 | 1/5 | • | P | 28 | 3 | 9.6 | • | | 1/16 | A, DB, H | +5, +12, -10 | 4.25×7 | Also called Super Jolt. 64 bytes of interrupt vector RAM on board. Microcomputer with keyboard, called VIM-1, also available. | 813 |
| Texas Instruments | TM 990/180M | 8/14 | TMS9980 | 2.5/3 | 16 | 0.5/1 | 2/4 | • | P | 24 | 1 | 38.4 | • | | 2/16 | A, DB, H | +5, ±12 | 11×7.5 | Serial data can be thru differential line driver/receiver. Prototyping area on board. | 814 |
| | TM 990/100M | 16/15 | TMS9900 | 3 | 32 | 0.5/1 | 2/4 | • | P | 16 | 1 | 38.4 | • | | 2/16 | A, DB, H | +5 ±12 | 11×7.5 | Multiply, divide, bit I/O instructions. Prototyping area on board. Another board also available. TM990/101M. | 815 |
| Western Digital | Microengine | 16/16 | WD-9000 | 3 | 64 | 64 | 0 | • | P | 16 | 2 | 19.2 | | | | H | ±5, ±12 | 8×16 | Executes programs directly from Pascal P-code without assembly or compiling. | 816 |
| Wintek | WINCE CMM | 8/16 | 6800 | 0.1/1 | 64 | 0.5 | .5/4 | • | P | 32 | 1 | 9.6 | | • | | A, DB, H | +5, ±12 | 4.5×6.5 | 54-pin/0.156 in. connectors. Some unusual accessory cards. | 817 |
| Xitan | The General | 8/16 | Z-80A | 2/4 | 1024 | 128 | 1/24 | • | S-100 | 3 ports | 0 | | • | • | 3/16 | A, DB, H, OS | ±5, ±12 | 12×16 | For use in smart terminal; can execute four tasks concurrently. | 818 |
| Zilog | Z80-MCB | 8/16 | Z-80 | 2.47 | 64 | 4/16 | 0/8 | • | P | 16 | 1 | 38.4 | • | • | 4/16 | A, AP, DB, H, OS | +5 | 7.7×7.5 | Board has a 126 pin interface bus. | 819 |

*provided/max. possible

N/A: not available