N° d'ordre : 352

UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir

LE TITRE DE DOCTEUR INGENIEUR

Mohamed REDJIMI

ETUDE ET REALISATION D'UN SYSTEME PARALLELE POUR LE TRAITEMENT GRAPHIQUE

Thèse soutenue le 25 septembre 1984 devant la Commission d'Examen

Membres du Jury	M.	DAUCHET	Président
	V.	CORDONNIER	Rapporteur
	M.	MERIAUX	Examinateur
	J.C.	GENTINA	Examinateur
	Mme	RUBAT du MERAC	Examinateur

P R O F E S S E U R S CLASSE EXCEPTIONNELLE

M.	CONSTANT Eugène	I.E.E.A.
M.	FOURET René	Physique
M.	GABILLARD Robert	I.E.E.A.
M.	MONTREUIL Jean	Biologie
M.	PARREAU Michel	Mathématiques
M.	TRIDOT Gabriel	Chimie
M.	VIVIER Emile	Biologie
M.	WERTHEIMER Raymond	Physique

P R O F E S S E U R S 1ère classe

M.	BACCHUS Pierre	Mathématiques
M.	BEAUFILS Jean-Pierre (dét.)	Chimie
M.	BIAYS Pierre	G.A.S.
M.	BILLARD Jean (dét.)	Physique
M.	BOILLY Bénoni	Biologie
M.	BOIS Pierre	Mathématiques
M.	BONNELLE Jean-Pierre	Chimie
M.	BOUGHON Pierre	Mathématiques
M.	BOURIQUET Robert	Biologie
M.	BREZINSKI Claude	I.E.E.A.
M.	CELET Paul	Sciences de la Terre
M.	CHAMLEY Hervé	Biologie
M.	COEURE Gérard	Mathématiques
M.	CORDONNIER Vincent	I.E.E.A.
M.	DEBOURSE Jean-Pierre	S.E.S.
M.	DYMENT Arthur	Mathématiques

PROFESSEURS 1ère classe (suite)

M. ESCAIG Bertrand	Physique
M. FAURE Robert	Mathématiques
M. FOCT Jacques	Chimie
M. GRANELLE Jean-Jacques	S.E.S.
M. GRUSON Laurent	Mathématiques
M. GUILLAUME Jean	Biologie
M. HECTOR Joseph	Mathématiques
M. LABLACHE COMBIER Alain	Chimie
M. LACOSTE Louis	Biologie
M. LAVEINE Jean Pierre	Sciences de la Terre
M. LEHMANN Daniel	Mathématiques
Mme LENOBLE Jacqueline	Physique
M. LHOMME Jean	Chimie
M. LOMBARD Jacques	S.E.S.
M. LOUCHEUX Claude	Chimie
M. LUCQUIN Michel	Chimie
M. MIGEON Michel Recteur à Grenoble	E.U.D.I.L.
M. MIGNOT Fulbert (dét.)	Mathématiques
M. PAQUET Jacques	Sciences de la Terre
M. PROUVOST Jean	Sciences de la Terre
M. ROUSSEAU Jean-Paul	Biologie
M. SALMER Georges	I.E.E.A.
M. SEGUIER Guy	I.E.E.A.
M. SIMON Michel	S.E.S.
M. STANKIEWICZ François	S.E.S.
M. TILLIEU Jacques	Physique
M. VIDAL Pierre	I.E.E.A.
M. ZEYTOUNIAN Radyadour	Mathématiques

P R O F E S S E U R S 2ème classe

M. ANTOINE Philippe	Mathématiques (Calais)
M. BART André	Biologie
Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mathématiques
M. BELLET Jean	Physique
M. BERZIN Robert	Mathématiques
M. BKOUCHE Rudolphe	Mathématiques
M. BODARD Marcel	Biologie
M. BOSQ Denis	Mathématiques
M. BRASSELET Jean-Paul	Mathématiques
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie
M. CARREZ Christian	I.E.E.A.
M. CAYATTE Jean-Louis	S.E.S.
M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean-Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
M. CROSNIER Yves	I.E.E.A.
M. CURGY Jean-Jacques	Biologie
Mle DACHARRY Monique	Géographie
M. DAUCHET Max	I.E.E.A.
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DELORME Robert	S.E.S.
M. DE MASSON D'AUTUME Antoine	S.E.S.
M. DEMUNTER Paul	C.U.E.E.P.

PROFESSEURS 2ème classe (Suite 1)

M. DENEL Jacques	I.E.E.A.
M. DE PARIS Jean-Claude	Mathématiques (Calais)
Mle DESSAUX Odile	Chimie
M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
Mme DHAINAUT Nicole	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean-Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique (Calais)
M. DUBUS Jean-Paul	I.E.E.A.
M. FAKIR Sabah	Mathématiques
M. FONTAINE Hubert	Physique
M. FOUQUART Yves	Physique
M. FRONTIER Serge	Biologie
M. GAMBLIN André	G.A.S.
M. GLORIEUX Pierre	Physique
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel (dét.)	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREGORY Pierre	I.P.A.
M. GREMY Jean-Paul	S.E.S.
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HENRY Jean-Pierre	E.U.D.I.L.
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JEAN Raymond	Biologie
M. JOFFRE Patrick	I.P.A.

PROFESSEURS 2ème classe (suite 2)

M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LANGRAND Claude	Mathématiques
M. LATTEUX Michel	I.E.E.A.
Mme LECLERCQ Ginette	Chimie
M. LEFEVRE Christian	Sciences de la Terre
Mle LEGRAND Denise	Mathématiques
Mle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques
M. LEMAIRE Jean	Physique
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	C.U.E.E.P.
M. LOUAGE Francis(dét.)	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
M. MESSELYN Jean	Physique
M. MESSERLIN Patrick	S.E.S.
M. MONTEL Marc	Physique
Mme MOUNIER Yvonne	Biologie
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RAOULT Jean François	Sciences de la Terre
M. RICHARD Alain	Biologie

PROFESSEURS 2ème Classe (suite 3)

M. RIETSCH François	E.U.D.I.L.
M. ROBINET Jean-Claude	E.U.D.I.L.
M. ROGALSKI Marc	Mathématiques
M. ROY Jean-Claude	Biologie
M. SCHAMPS Joël	Physique
Mme SCHWARZBACH Yvette	Mathématiques
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STAROSWIECKI Marcel	E.U.D.I.L.
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole
Mme TJOTTA Jacqueline (dét.)	Mathématiques
M. TOULOTTE Jean-Marc	I.E.E.A.
M. TURRELL Georges	Chimie
M. VANDORPE Bernard	E.U.D.I.L.
M. VAST Pierre	Chimie
M. VERBERT André	Biologie
M. VERNET Philippe	Biologie
M. WALLART Francis	Chimie
M. WARTEL Michel	Chimie
M. WATERLOT Michel	Sciences de la Terre
Mme ZINN JUSTIN Nicole	Mathématiques

CHARGES DE COURS

M. ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M. BAFCOP Joël	I.P.A.
M. DUVEAU Jacques	S.E.S.
M. HOFLACK Jean	I.P.A.
M. LATOUCHE Serge	S.E.S.
M. MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M. NAVARRE Christian	I.P.A.
M. OPIGEZ Philippe	S.E.S.

A Papa et Maman,

A Aissa et Brahim,

A Christine.

REMERCIEMENTS

Je tiens à remercier

Monsieur M. DAUCHET, Professeur à l'Université de LILLE 1 qui m'a fait l'honneur de présider le Jury de cette Thèse.

Monsieur V. CORDONNIER, Professeur à l'Université de LILLE 1, qui a accepté de diriger ma Thèse, ainsi que pour ses encouragements et critiques.

Monsieur M. MERIAUX, chargé de recherche au C.N.R.S., avec lequel, j'ai eu des discussions intéressantes et constructives notamment dans le domaine du graphique.

Monsieur J.C. GENTINA, Professeur à l'I.D.N., qui m'a fait l'honneur d'examiner cette Thèse.

Madame C. RUBAT DU MERAC, Ingénieur au C.I.M.E. de Grenoble, qui a accepté de participer au Jury de Thèse.

Mes collègues du Laboratoire pour les différents échanges de points de vue que nous avions eu et notamment les membres de l'équipe graphique.

Ma famille, qui malgré la distance qui nous sépare, n'a cessé d'encourager cette initiative.

Le Secrétariat du Laboratoire et particulièrement Madame B. VANDROEMME, qui a réalisé la frappe de ce document avec soin et gentillesse. Ainsi que Madame H. DEBOCK qui en a consciencieusement assuré l'impression.

ETUDE ET REALISATION D'UN SYSTEME PARALLELE POUR LE TRAITEMENT GRAPHIQUE

M. REDJIMI
UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE I
U.E.R. d'I.E.E.A. - Informatique
59655 VILLENEUVE D'ASCQ CEDEX

TABLE DES MATIERES :

0. INTRODUCTION GENERALE

Nous assistons, ces dernières années, a une explosion de l'informatique graphique. En effet quoi de plus normal que de s'intéresser aux images ! Il est largement admis que plus de 90% de nos connaissances nous sont acquises grâce à notre système visuel (lecture, cinéma, télévision, images diverses...). De plus une image permet une observation globale, compacte et non ambigüe (le plus souvent) d'un phénomène tout en offrant une multitude d'informations.

L'interprétation de certaines images peut différer d'un individu à un autre si ces images comportent des éléments prêtant à ambiguité (flous, symboliques, peinture...) ou si les observateurs sont de cultures différentes voire parfois d'humeurs différentes. Ceci fait ressortir le caractère subjectif que peut refléter une image.

Historiquement, l'informatique graphique était réservée surtout pour des applications spécialisées telles que la C.A.O. (Conception Assistée par Ordinateur) par exemple, domaine où l'homme s'allie l'ordinateur à des fins de production, qui était donc une affaire de spécialiste.

Actuellement, les différents domaines de la X.A.O*. (C.A.O., D.A.O., E.A.O., G.P.A.O....) bénéficient de plus en plus de l'application de l'informatique en général et du graphique en particulier. Cependant, on assiste en parallèle à une pénétration globale dans les domaines grands publics par l'intermédiaire des micro-ordinateurs offrant entre-autres divers jeux vidéos, ainsi que par le biais de la télévision et du cinéma où bon nombre de films utilisent l'ordinateur pour synthétiser des images et créer des effets spéciaux.

Tout ceci a fait se développer un marché potentiel et ouvert des voies de recherche.

---

* C.A.O. = Conception Assistée par ordinateur.
D.A.O. = Dessin Assisté par Ordinateur.
E.A.O. = Enseignement Assisté par Ordinateur.
G.P.A.O. = Gestion de Production Assistée par Ordinateur.
I.A.O. = Illustration assistée par Ordinateur.

Nous pouvons distinguer deux catégories de systèmes graphiques :

- Les systèmes de traitement d'images.
- Les systèmes de synthèse d'images.

Dans le premier cas, le système reçoit des images sous forme codée et effectue dessus des traitements consistant le plus souvent en :

- Extraction de primitives.
- Reconnaissance d'images.
- Etc...

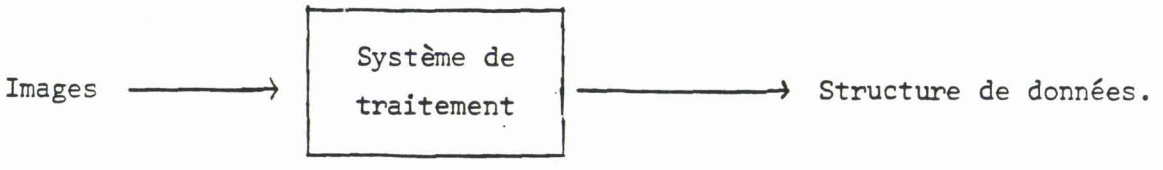


Fig. 0.1. : Système de traitement d'images.

Dans le second cas, les différents organes du système graphique oeuvrent dans le but de synthétiser et de produire des images. On peut alors considérer deux sortes de systèmes :

a - Les systèmes pour lesquels l'image constitue un support d'information :

Dans ce cas l'image obtenue ne constitue pas la finalité du système, l'intérêt est reporté sur les informations qu'elle représente. Ceci s'applique par exemple, dans certains cas où l'image sert d'outil pédagogique : représentation d'objets destinés à illustrer un cours... ainsi qu'en C.A.O. : maquettes de prototypes, etc...

b - Les systèmes pour lesquels l'image constitue une finalité :

L'image obtenue doit être la plus réaliste et la plus nette possible. Plusieurs domaines d'application nécessitent de telles images :

- Cinéma.
- Simulateurs de vol.
- Arts plastiques
- Etc...

Un système de synthèse peut se représenter de la façon suivante :

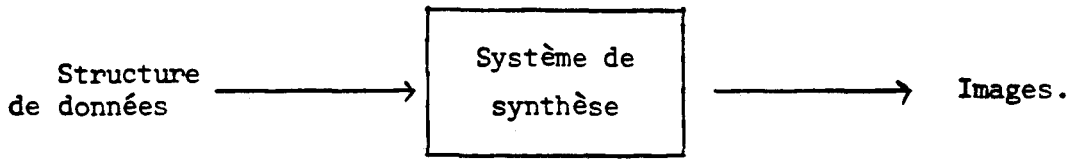


Fig. 0.2. : Système de synthèse d'images.

Nous pouvons considérer aussi une troisième catégorie des systèmes graphiques où le système reçoit un doublet <Images, Structure de données> et fournit en sortie le même doublet dont les éléments auraient subi eventuellement des modifications.

C'est le cas par exemple des systèmes de manipulation d'images qui reçoivent en entrée une image et qui effectuent diverses opérations dessus (transformations géométriques, opérations arithmétiques et logiques avec d'autres images...) pour fournir en sortie une nouvelle image.

Cette distinction ne considère que les aspects fonctionnels de ces systèmes et elle est incomplète car elle ne prend pas en compte une composante très importante dans la majorité des cas qui est le temps. En effet un système de traitement d'images, par exemple, peut être caractérisé par son temps de réponse si l'apparition d'une image i à son entrée doit déclencher un processus ρ dans un délai Δt c'est le cas des systèmes de traitement d'images en temps réel où le temps de réponse du système revêt une importance considérable.

La composante temps est très importante aussi dans le cas de systèmes de synthèse et d'animation d'images si ces images sont destinées a être consommées au fur et à mesure de leur production par un oeil humain par exemple. Si ces images sont affichées sur un écran de télévision, la fréquence de leur production doit être voisine de 25 images par seconde, ce qui impose un rythme de 40 ms par image synthétisée. En pratique, on distingue généralement deux grandes catégories de systèmes :

- Les systèmes produisant des images en différé ; l'image est dans ce cas produite au rythme du calculateur. Son affichage a lieu dans un deuxième temps à partir d'une unité de stockage rapide. Ce procédé est très utilisé dans les systèmes d'animation d'image.

- Les systèmes produisant des images en temps réel ; dans ce cas l'image est produite au rythme de sa consommation. Ce procédé implique un fonctionnement très rapide du calculateur. Dans le cas, toujours, où les images sont visualisées sur une télévision une image doit être synthétisée toutes les 40 ms.

Le système que nous proposons rentre dans cette deuxième catégorie.

Le travail présenté se décompose de la façon suivante :

- Dans le Chapitre I, nous abordons les concepts généraux liés à la synthèse d'images réalistes.

- Le Chapitre II décrit l'architecture de l'élément central intervenant dans le système graphique que nous proposons (le processeur graphique).

- Le Chapitre III décrit l'environnement du système graphique proposé caractérisant ainsi le système de communication avec l'utilisateur (processeur hôte, unités de communication et de visualisation...).

- Le Chapitre IV dégage quelques éléments essentiels du logiciel de base offrant un noyau de synthèse et de manipulation d'images.

- Le Chapitre V, enfin, décrit l'interface fonctionnelle avec l'usager qui consiste en l'implémentation d'un langage de haut niveau ; le langage FORTH.

SOMMAIRE DU CHAPITRE I

I. PRINCIPES GENERAUX

Ce chapitre tente de faire une synthèse des principes de base de l'informatique graphique. Signalons à cet effet l'existence d'une bibliographie très riche (MOL 76, NES 79, FOV 82).

I.1. Introduction

Un système graphique peut être défini comme étant l'ensemble des moyens informatiques mis en oeuvre afin d'assurer le traitement, le stockage et la visualisation des informations concernant les "images" et les "dessins" appelées aussi informations graphiques. Cette appelation est prise ici dans son sens le plus large, car une "image" est définie sous des formes codées différentes depuis sa création (par l'intermédiaire d'un clavier alpha-numérique par exemple) puis son passage par les différentes chaînes de traitement (sous d'autre formes codées éventuellement) jusqu'à sa visualisation (sous formes d'intensités lumineuses sur un écran de télévision par exemple).

I.2. Notion de système de synthèse d'images

La configuration d'un système de synthèse d'images peut être vue de la façon suivante : (MER 79, MOL 76)

- Le terminal assurant la visualisation des images.
- Un processeur hôte.
- Un ensemble de logiciels permettant de gérer les programmes utilisateurs et d'offrir des fonctions et des primitives de programmation (noyau de systèmes, langages de haut niveau...).
- Un ensemble d'unités de stockage dans le but de mémoriser les informations du système graphique (les logiciels, les structures de données nécessaires à ces logiciels, les bases de données graphiques...).
- Un ensemble d'unités de dialogue dont le rôle est d'assurer la communication entre l'utilisateur et le système graphique.

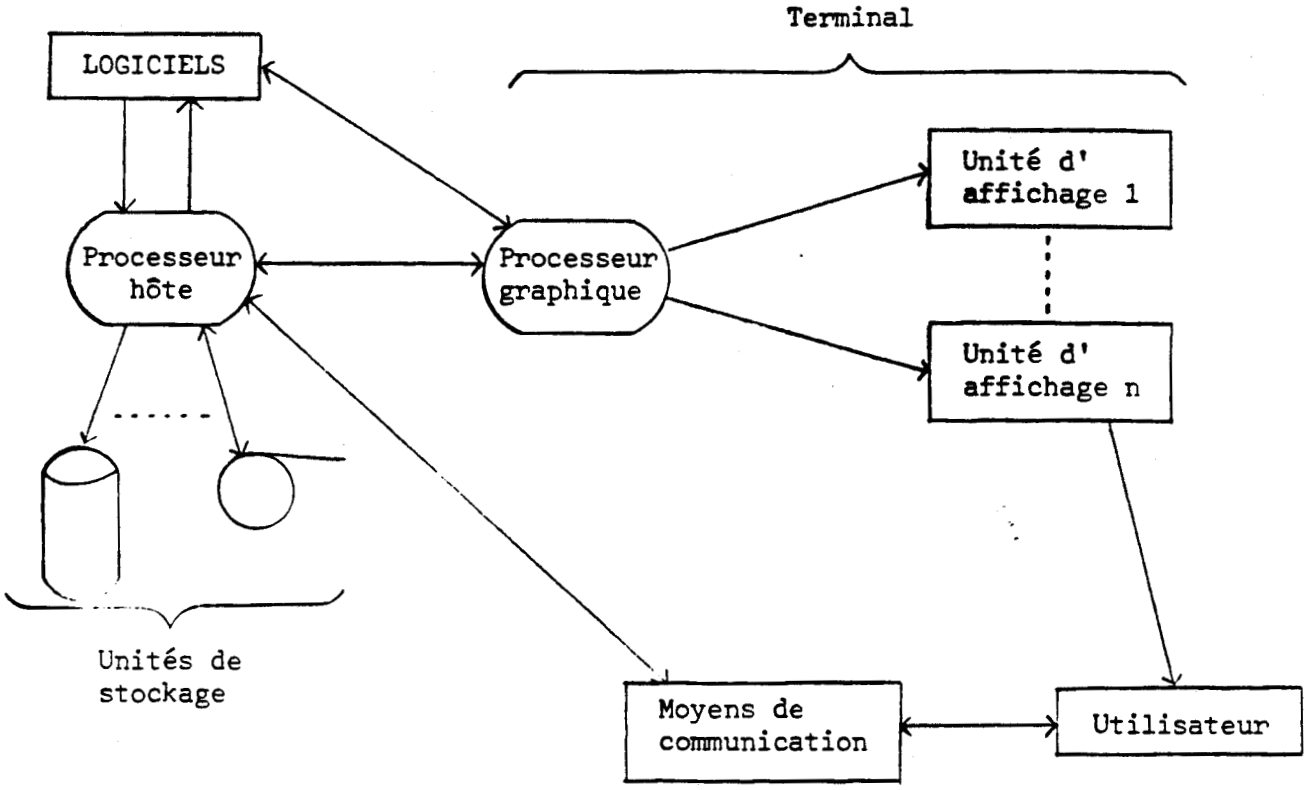


Fig. : I.1. : Architecture générale d'un système graphique intéractif.

Dans le cas où l'utilisateur peut dialoguer avec le système graphique au fur et à mesure de l'exécution de son programme influant ainsi sur le déroulement interne de celui-ci ; le système est dit intéractif. Dans ce cas il y a "FEED-BACK" entre l'utilisateur et le système. Ceci implique des caractéristiques particulières au plan matériel : outils adaptés de saisie et logiciel : fonctions de saisie, de mise à jour...

1.2.1. Le terminal

Le terminal est en fait composé des unités d'affichage et d'un processeur graphique.

1.2.1.1. Les unités d'affichage

Ce sont les moyens physiques d'affichage des images ou dessins. Ce sont les imprimantes, les tables traçantes, les écrans à tubes cathodiques... nous nous intéresserons à ces derniers.

a - Caractéristiques techniques (MOL 76, VAN)

- Dimension de l'écran : La dimension de l'écran affecte la quantité d'informations affichables sur l'écran. Elle est définie par la diagonale habituellement de 40 à 60 cm (25 × 25 à 40 × 30 cm) pour les écrans cathodiques.

- Définition : Détermine l'écart minimum entre deux entités affichées et limite ainsi la quantité d'informations visualisables sur l'écran pour une dimension donnée.

  - Définition technologique : liée essentiellement à la tâche lumineuse élémentaire.

  - Définition informatique : détermine le nombre maximum de points adressables de l'écran (soit 256 × 256, 512 × 512, 1024 × 1024, 100.000 × 70.000

- Précision : C'est l'écart entre la position programmée et la position affichée. Constante pour les écrans à éléments d'affichage discrets (cellules, diodes...) ; elle est variable pour les écrans à rayons cathodiques.

- Fidélité : Permet de désigner sur l'écran le même point pour les mêmes coordonnées.

- Brillance et contraste : Assurent des affichages nets.

b - Généralités sur la couleur

Pour certaines applications, deux niveaux d'intensité suffisent pour représenter l'image désirée (on parle alors d'images bi-niveaux ou bit-map).

D'autres applications nécessitent beaucoup plus de niveaux d'intensités (ou de niveau de gris) c'est le cas des images en "noir et blanc" où les niveaux de gris s'échelonnent entre la valeur d'intensité minimale correspondant au noir et la valeur d'intensité maximale correspondant, quant à elle, au blanc. Les intensités comprises entre ces deux valeurs déterminent chacune un niveau de gris.

Cependant, pour obtenir des images réalistes ceci ne suffit pas ; une contrainte essentielle est que ces images soient colorées. D'après les principes de la trichromie une couleur quelconque peut être obtenue par mélange des trois couleurs de base (Rouge, vert et bleu). Ainsi une couleur quelconque c peut être représentée comme la somme de proportions de rouge, de vert et de bleu :

$$||s(C) = r(R) + v(V) + b(B)$$

s : intensité de la couleur C.

r(R), v(V), b(B) intensité des couleurs primaires Rouge, Vert et Bleu entrant dans la composition de C.

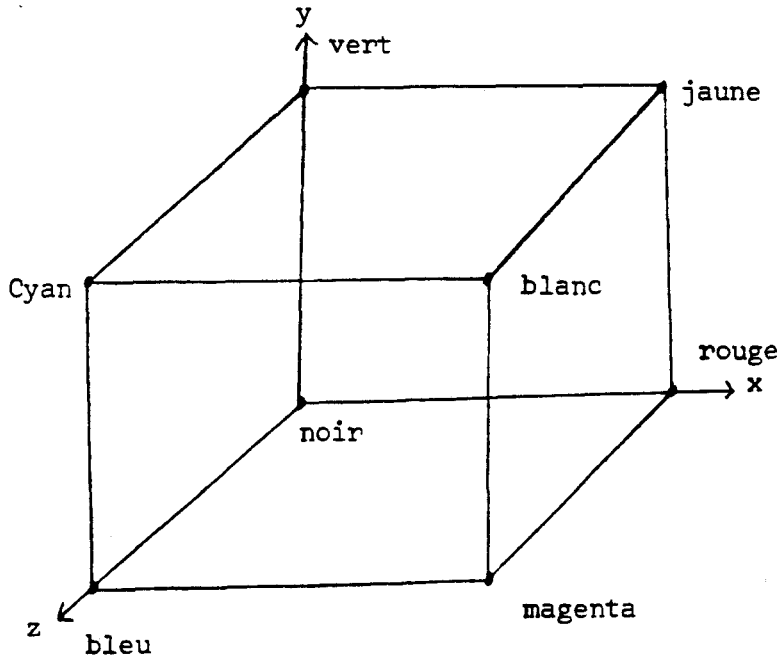


Fig. I.2. : Cube des couleurs.

Exemple :

Rouge	Vert	Bleu	Couleur obtenue
0	0	0	noir
0	0	1	bleu
0	1	0	vert
1	0	0	rouge
1	1	0	jaune
1	1	1	blanc
1	0	1	magenta
0	1	1	cyan

"0" : Absence de la couleur.
"1" : Présence de la couleur.

Il n'est pas facile, cependant, de prévoir les proportions de rouge, vert et bleu pour obtenir une couleur donnée ou, à l'inverse, de déterminer la couleur qui va correspondre à un mélange donné. Cette raison a conduit à définir différents modèles facilitant la description des couleurs, parmi eux les modèles T.I.S. et T.B.N. (MER 82, FER 83).

c - Notions sur le téléviseur couleur (MER 79, MOL 76)

Plusieurs systèmes existent, nous présentons le tube à masque (modèle PIL) qui a servi de support à ce travail.

Ce tube est formé d'une surface parsemée de triplets de points (appelés triades) émettant dans les trois couleurs de base le rouge, le vert et le bleu. Toutes les autres couleurs peuvent être obtenues en combinant du rouge, du vert et du bleu. Trois canons à électrons (un par couleur de base) permettent d'agir sur les triades provoquant ainsi les émissions de lumière. Enfin un masque perforé permet de guider les faisceaux vers les points considérés (Fig. I.5.).

Sur le tube utilisé, chaque point a un diamètre de 0,42 mm environ, la distance entre deux points voisins est de 0,72 mm environ. L'écran fait 54 cm et comporte donc environ 357.000 triades.

Nous disposons actuellement d'autres moniteurs ayant un écran de 51 cm, la distance entre deux points étant d'environ 0,3 mm soit une surface de 1,2 millions de triplets environ.

d - Génération et entretien de l'image

Les images sont dessinées sur l'écran par déplacement du faisceau d'électrons et par le contrôle de son intensité. La courte persistance de l'effet luminescent impose ; pour rendre l'image permanente ; de la rafraîchir (25 fois par seconde environ). Il existe deux modes de balayage : Le balayage cavalier et le balayage télévision.

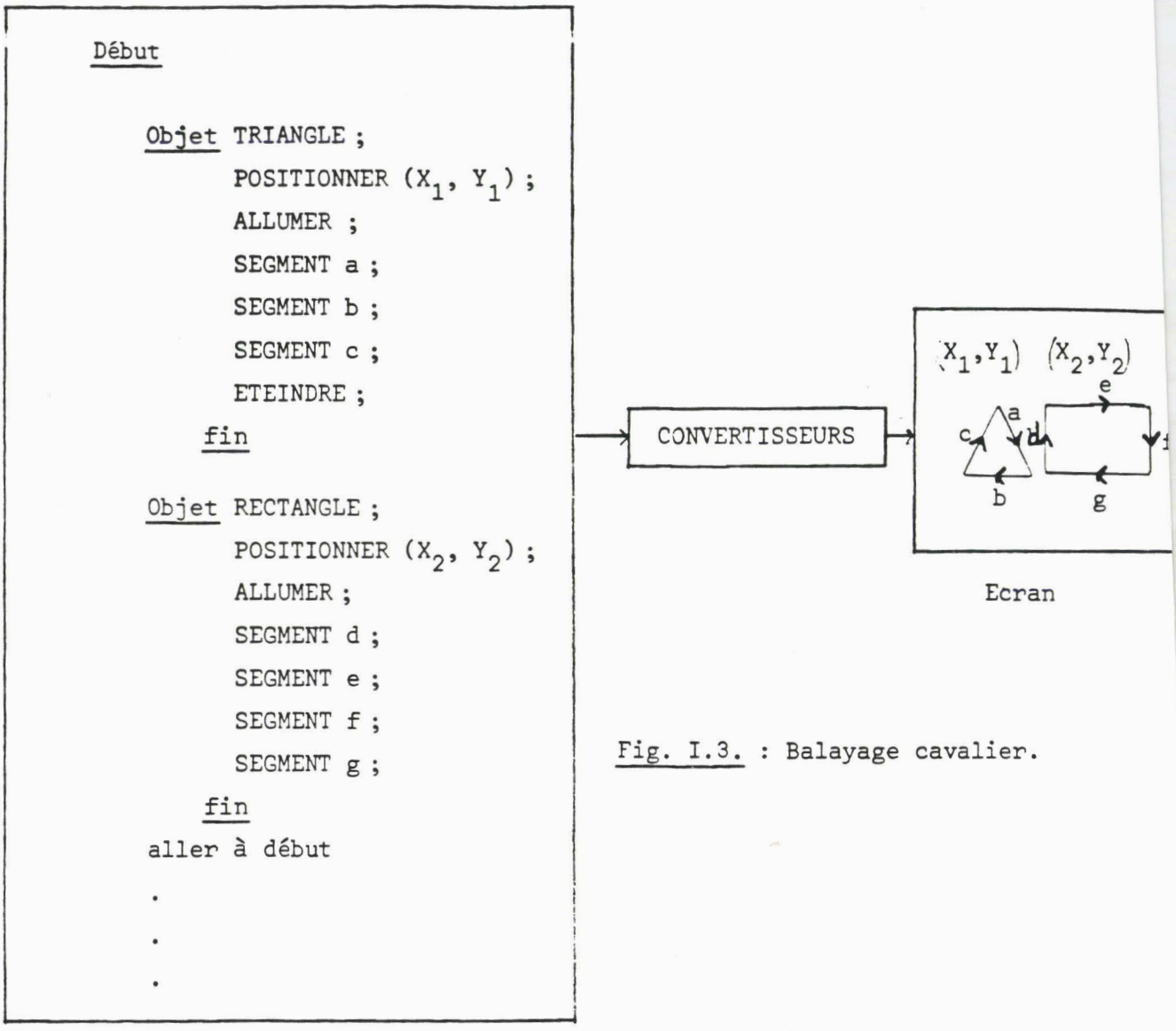
- Le balayage cavalier : Le faisceau d'électrons décrit un parcours correspondant à l'image affichée. Les ordres graphiques nécessaires à la gestion de l'écran sont des types suivants :

- Positionnement du spot en un point (x, y) de l'écran.
- Allumage du spot.
- Tracer d'un segment depuis le point (X_1, Y_1) au point (X_2, Y_2)
- Extinction du spot.

En évitant que le spot ne reste allumé beaucoup de temps en un seul point de l'écran au risque d'abîmer ce dernier.

Le processeur gère, dans ce cas, une liste de visualisation correspondant aux différents objets à visualiser. Les performances du système sont, alors, fonction de la longueur de cette liste.

Exemple :



Liste de visualisation

Fig. I.3. : Balayage cavalier.

- Le balayage télévision : Le balayage de l'écran est systématique et ne dépend plus de la liste de visualisation. Il est dit aussi balayage de trame. Le spot balaye l'ensemble de l'écran de la gauche vers la droite et de haut en bas en une trame. En fait pour éviter un phénomène très désagréable de scintillement, le balayage se fait en deux demi-trames l'une concernant les lignes paires, la seconde les lignes impaires; c'est le balayage entrelacé.

Dans le standard européen : (voir Chapitre III).

- Trames de 625 lignes.
- Affichage d'une ligne toutes les 64 µs donc une trame dure 40 ms.

- Affichage entrelacé de deux demi-trames de 312,5 lignes chacune.

Dans ce cas les pixels sont affichés au fur et à mesure de leur présentation. Généralement une mémoire d'entretien (ou mémoire de trame) contient la configuration de l'image à afficher.

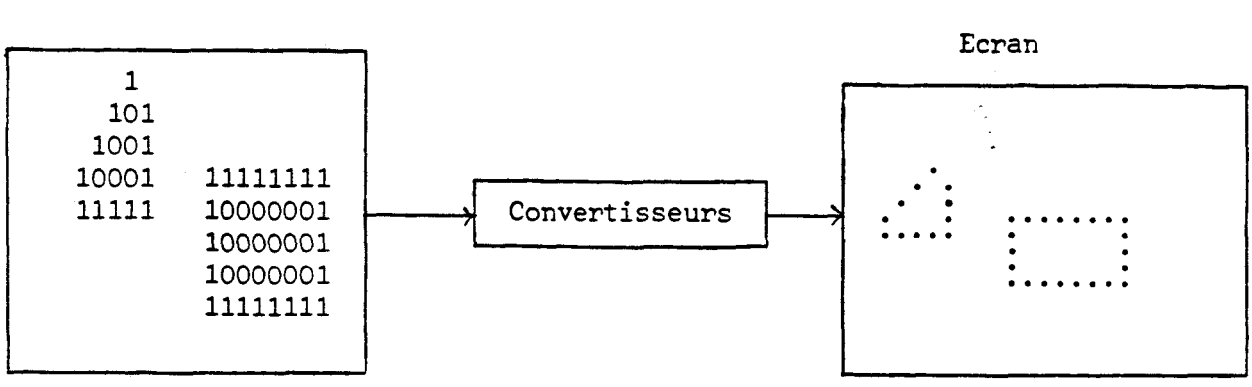


Fig. I.4. : Exemple de configuration pour un écran de télévision bi-niveaux (bit-map).

Conclusion : Les avantages du balayage télévision sont nombreux :

- Bonne stabilité de l'image vue son indépendance vis-à-vis des éléments à afficher.

- Faible coût (télévision domestique).

- Plusieurs niveaux de gris et couleurs peuvent être utilisés.

Le principal inconvénient est dû au fait que la capacité de la mémoire de trame devient importante pour une résolution élevée de l'écran.

1.2.1.2. Le processeur graphique

C'est la partie intelligente du terminal. Son architecture est de complexité variable selon la configuration du système. Son rôle est de :

- Interpréter les commandes issues des unités de traitement et de l'utilisateur et agir en conséquence sur l'image à visualiser.

- Gérer la liste de visualisation dans le cas d'un affichage en mode cavalier. Gérer la mémoire de trame dans le cas d'un balayage télévision...

- Dans certains cas c'est le processeur graphique qui s'occupe des transferts proprement dit entre les mémoires de rafraichissements et les unités d'affichage (Initialisation de programmes canaux assurant la sortie sur table traçante...).

Il est donc nécessaire que le processeur graphique soit très rapide, performant et spécialisé, c'est l'une des raisons pour lesquelles plusieurs applications actuelles utilisent des processeurs microprogrammables. (LABS 78, MOL 76).

1.2.2. Le processeur hôte

Son rôle est de décharger le processeur graphique de tâches informatiques plus classiques telles que la manipulation de grands ensembles de traitement (gestion de bases de données, compilation, gestion de fichiers...).

On trouve souvent de gros ordinateurs (CRAY 1...) locaux au système graphique ou éloignés, dans ce cas une ligne de télétransmission (VIA un réseau par exemple) assure la correspondance.

Dans plusieurs applications clés en main et surtout avec les diminutions des coûts du matériel dûs à la révolution des circuits intégrés LSI et VLSI, les processeurs hôtes sont basés autour de cartes micro-ordinateurs, offrant la possibilité de réaliser des connexions avec de grosses machines et un fonctionnement autonome.

1.2.3. Les unités de stockage

Ce sont les mémoires de masse du système ; ils jouent le même rôle que dans les systèmes informatiques classiques c'est-à-dire la mémorisation d'ensembles importants d'informations (bibliothèques de programmes, compilateurs, bases de données,...).

On y trouve les disques dont les capacités actuelles avoisinent les 500 Moctets, les bandes magnétiques, les disquettes... Notons l'apparition sur le marché des vidéodisques permettant le stockage de capacités très importantes d'informations ainsi que les bandes magnétiques vidéos...

1.2.4. Les unités de communication (MOL 76, MER 79)

Un système graphique intéractif doit satisfaire les quatre fonctions de base :

- Le menu : Permet à l'opérateur de choisir une action à effectuer parmi une liste de commandes.

- L'introduction de valeurs permettant de former les éléments du dialogue.

- L'identification d'un ou plusieurs composants d'une image permettant ainsi de préciser le ou les éléments concernés par un traitement spécifique.

- L'acquisition d'un élément d'image dans le but de construire une représentation destinée au calculateur.

De façon plus concise, ces unités permettent le dialogue par l'intermédiaire de fonctions de programmation et de commande (commandes de chargement et d'exécution de programmes, mise au point...) et de transmissions de données (transmission de coordonnées...).

Matériellement, ces systèmes sont :

- Des moyens d'entrées-sorties classiques : claviers alpha-numériques, imprimantes, unités de disquettes...

- Des moyens spécifiques à l'informatique graphique. Nous en citerons quelques-uns ; la liste n'étant pas exhaustive :

a - Stylo, photosyle

Utilisés avec les écrans rafraichis ; ils détectent le passage du spot lors du balayage et fournissent ses coordonnées ou permettent d'itentifier un objet

b - Manche à balai, boule roulante

Associés à un marqueur sur l'écran (flèche ou réticule) permettent de désigner un point sur celui-ci.

c - Clavier de fonctions

C'est un boîtier de touches relié au système graphique chaque touche peut être programmée pour réaliser une fonction spécifique.

d - Tablette à numériser, digitaliseur

Transmet au processeur les coordonnées du "crayon" avec lequel on dessine Ces outils sont utilisés pour numériser des documents (2D) ou des pièces (3D).

1.2.5. Les logiciels graphiques

Ce vocable regroupe toutes les fonctions programmées du système. Les systèmes de synthèse d'image existant peuvent se classer en deux grandes catégories :

- Les systèmes à vocation universelle (ou systèmes généraux) permettent l'utilisation de matériels divers. Leur utilisation est possible dans différents domaines application.

- Les systèmes spécialisés destinés à une application spécifique et fonctionnant sur un type déterminé de matériel.

Les systèmes généraux tirent leur puissance de leur universalité. Leur rendement ne peut, cependant, pas être optimal quant à leur application dans un domaine spécial nécessitant un certain ordre de puissance et de performances ainsi qu'une configuration spécifique.

A l'inverse, les systèmes spécialisés ont un rendement optimal dans le domaine pour lequel ils ont été conçus et réalisés et présentent de lourds handicaps (sinon une impossibilité) quant à d'autres applications.

Cette diversification ne permet cependant pas une normalisation facile au contraire des systèmes généraux.

Nous présenterons deux structures différentes de logiciels généraux. La première est issue d'une volonté de normalisation (le modèle utilisé par le GSPC core system), la seconde correspond au modèle défini par LUCAS.

1.2.5.1. Le modèle utilisé par le système GSPC (GSP 79, HER 79)

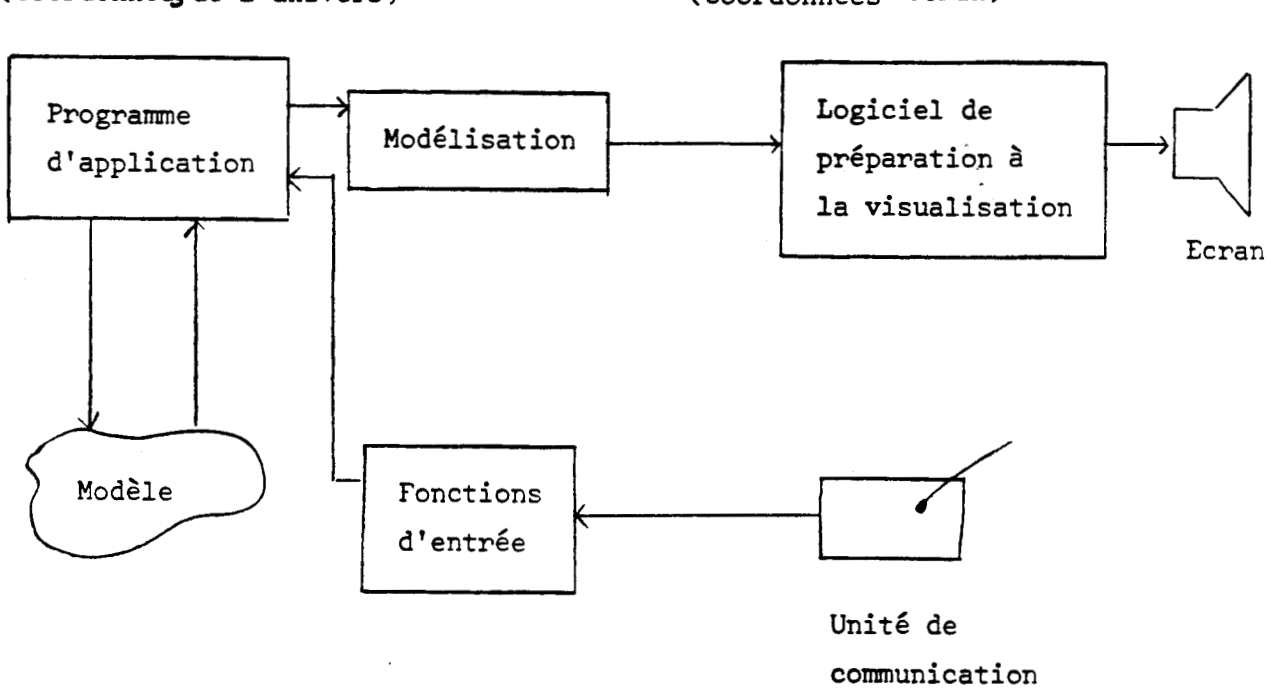


Fig. I.5. : Structure du modèle utilisé par le système GSPC.

Nous pouvons faire les remarques suivantes au sujet de ce modèle :

- La séparation entre les fonctions d'entrée et de sortie.
- Le concept de deux systèmes de coordonnées différents. Une image peut être définie dans le système de coordonnées de l'univers (en 3D le plus souvent) puis traduite dans celui de l'écran (en 2D, possibilité de fenêtrage) grâce au logiciel de préparation à la visualisation.

La portabilité ainsi que l'indépendance par rapport aux périphériques sont assurées.

1.2.5.2. Le modèle de LUCAS (LUC 77, CEI 79, KIL 81)

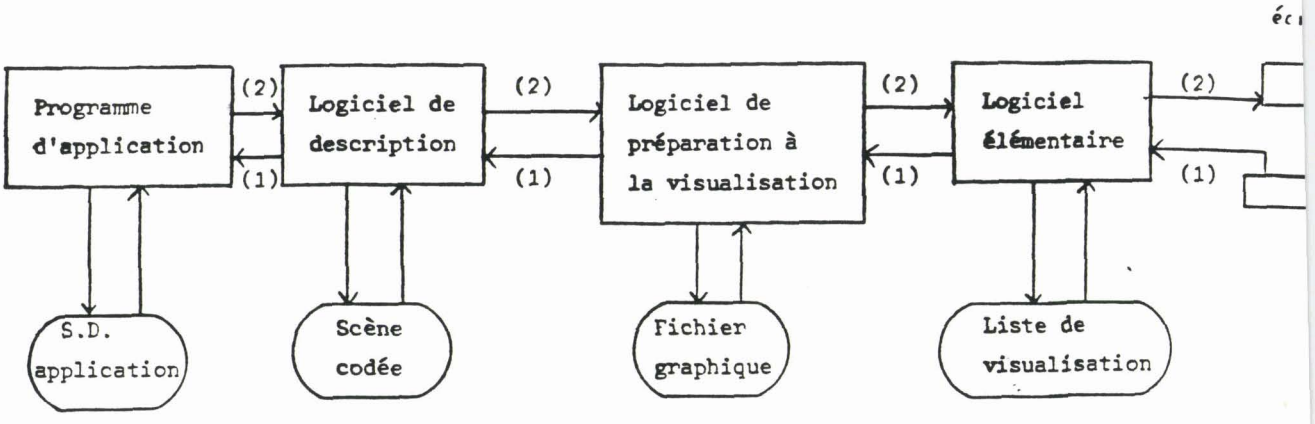


Fig. I.6. : Schéma du modèle en couches

non nécessaire à tous les terminaux

Ce modèle met en évidence les différentes intéractions existantes au niveau des logiciels.

Chacun de ces logiciels comprend trois parties :

- Une bibliothèque d'algorithmes.
- Une structure de données.
- Un moniteur d'enchaînement (ou interpréteur).

Le "chemin" (2) correspond à la synthèse, le "chemin" (1) à l'intéraction

Ce modèle permet les mêmes fonctions que le précédent et le complète en quelque sorte.

I.3. Concepts liés à la synthèse d'images réalistes

Le processus de production d'images réalistes peut se subdiviser en deux étapes distinctes. En premier lieu ; il est nécessaire de composer les différents éléments constituant la scène ; en 3D le plus souvent. Cette étape correspond à la formation d'une maquatte numérique.

Dans un deuxième temps cette maquette est analysée en fonction d'un observateur de manière à produire une "vue" de celle-ci destinée à être affichée sur un support à deux dimensions. Plusieurs aspects sont donc à prendre en compte dans un tel processus. (MAR 82).

- Les informations concernant la structure des objets composant la scène, celles-ci expriment les relations entre ces objets (un objet peut être composé de plusieurs autres plus simples...).

- Les informations correspondant à la morphologie des objets exprimant ainsi la forme de ceux-ci.

- Les informations d'aspects définissant la couleur, la brillance, la transparence...

- Les informations d'éclairage exprimant les sources lumineuses éclairant ces objets.

- Les informations géométriques permettant de situer les objets les uns par rapport aux autres dans un repère donné.

Ces informations permettent de caractériser les objets composant la scène (maquette numérique), le traitement suivant permet de déterminer les incidences entre les objets aboutissant ainsi à déterminer les ombres portées et la coloration des objets en fonction les uns des autres (BLI 77, WHI 80, OLE 82). La maquette étant destinée à être visualisée sur un dispositif à ^{2} dimensions, il s'agit de déterminer les surfaces visibles sur ce dispositif, ce problème se pose en trois termes :

a - Choix du point d'observation de la scène déterminant une "vue" de celle-ci.

b - Passage d'un espace en 3D en un espace en 2D.

c - Clôturation de l'espace d'arrivée (en effet le plan obtenu après le point b peut être plus grand que l'espace de visualisation).

Plusieurs algorithmes ont été proposés pour traiter de cette question, nous en décrirons deux :

- L'algorithme de WATKINS (MOL 76) publié en 1970 ; cet algorithme se fonde sur la notion de lignes de balayage et est destiné à un affichage sur un écran

de télévision. La scène est balayée selon un balayage de trame (de haut en bas et de gauche à droite). Chaque ligne de balayage produit un ensemble de segments horizontaux (la scène est coupée selon un plan π parallèle à l'axe OX de l'écran. Une analyse de visibilité permet de dégager les segments à visualiser. Cet algorithme a été cablé et permet le traitement de scènes complexes en temps réel.

- Un second algorithme, plus efficace a été présenté en 1972 par NEWELL, NEWELL et SANCHA (NNS 72) le principe de cet algorithme suppose la numérotation des faces selon leur éloignement de l'observateur. La visualisation se fait en commençant par la face la plus éloignée de l'observateur, on peint ensuite la suivante qui vient cacher totalement ou partiellement ou se combine avec la première (cas de la transparence) etc...

D'autres algorithmes ont été développés sur ce thème, on trouvera une étude approfondie de la question dans (LUC 77, BOU 80).

I.4. Etude de quelques machines de synthèse d'images

Traditionnellement, l'un des domaines d'application des systèmes de synthèse d'images est la conception assistée par ordinateur (C.A.O.). Ce terme regroupe toutes les applications où il y a collaboration entre l'homme et l'ordinateur pour l'étude, la définition et la conception d'un produit (VAN). Des outils logiciels et matériels spécifiques à plusieurs domaines de C.A.O. ont été développés (MIC 80) et des produits clés en main existent actuellement sur le marché (conception de circuits imprimés, de carrosseries de voitures,...). Cependant, l'application de ces systèmes ne s'est pas arrêtée là, actuellement d'autres domaines très différents les uns des autres en bénéficient :

- Jeux vidéos
- Cinéma
- Enseignement assisté par ordinateur (E.A.O.)
- Arts graphiques (peinture, dessin,...)
- Simulation (simulateurs de vol,...)
- etc...

Les moyens mis en oeuvres dans ces différentes applications ne sont évidemment pas les mêmes, une configuration pour des jeux vidéos, par exemple, est moins complexe en capacités de traitement et de stockage que celle qui se propose la simulation.

1.4.1. Etude de quelques matériels de synthèse d'images

1- Le TEKTRONIX 4027 (MIC 80, MAR 82).

Ce terminal est basé autour de microprocesseurs gérant des images couleurs. Les opérateurs de synthèse permettent notamment :

- La génération de points, vecteurs,... sous diverses textures (pointillé, trait mixte, tirets, etc,...).

- Le remplissage de tâches à l'aide de "motifs", sortes de textures simplifiées comportant 14 × 8 points de 2 couleurs choisies parmi 64.

Nous trouvons en option de ce système :

- 192 K de mémoire graphique.
- Interface CCITT.
- Reprographe.
- Possibilité de ROM additionnable.

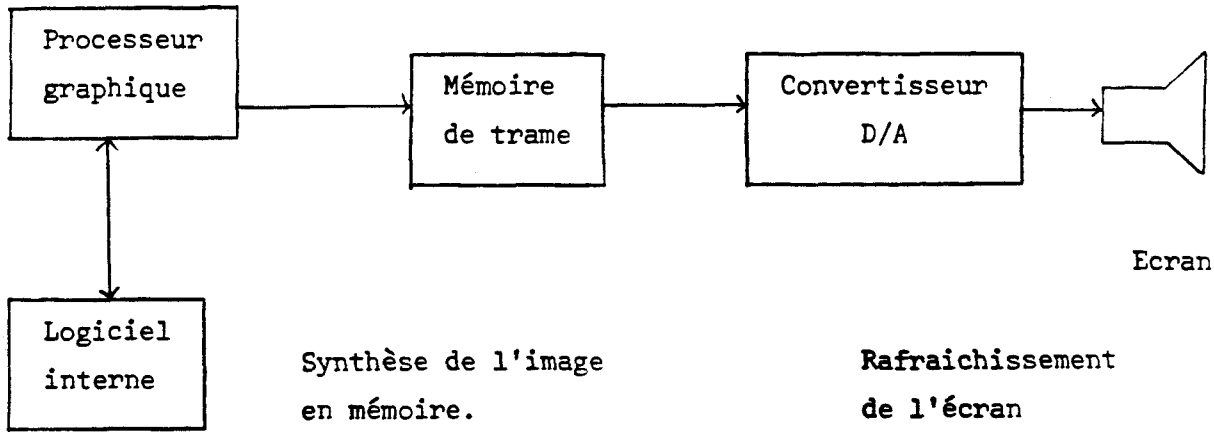


Fig. I.7. : Le TEKTRONIX 4027.

Citons aussi les ordinateurs graphiques de table TEKTRONIX 4052 et 4054 dont les principes sont sensiblement les mêmes.

2- Le système RAMTEK RM-9460 (GOC 82)

Ce système est à base de quatre microprocesseurs en tranche 2901 permettant de gérer en parallèle des mémoires d'images. Un 5ème processeur Z80 permet l'interface avec l'extérieur. l'architecture du système est schématisée ci-dessous :

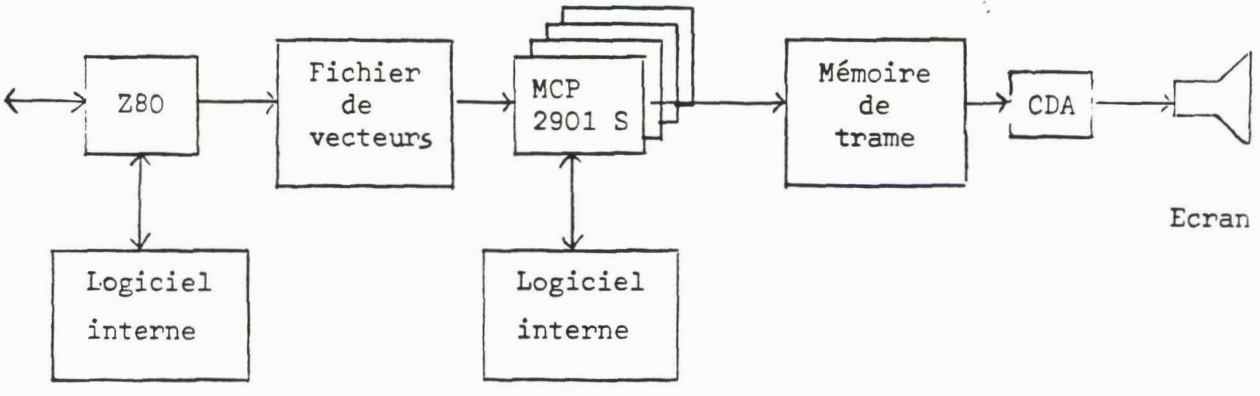


Fig. I.8. : Le système RAMTEK RM 9460.

3- Le système de simulation de vol VISA de THOMSON-CSF (ALL 83)

Ce système a une capacité d'affichage de 1000 faces en temps réel (extensible). L'association avec une base de données dynamique permet la visualisation de scènes pouvant contenir plusieurs milliers de facettes.

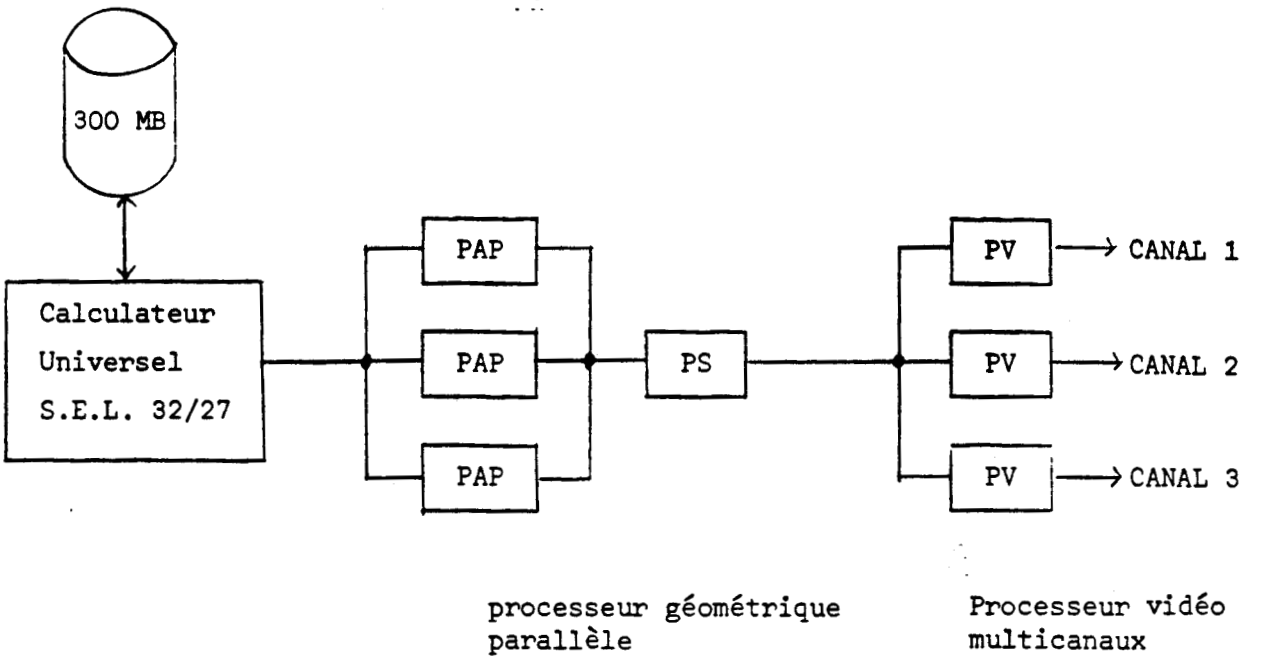


Fig. I.9. : Organisation du système VISA.

PAP : Processeur arithmétique programmable.

PS : Processeur de segment.

PV : Processeur vidéo.

4- Le projet HELIOS (FER 81, MAR 82, MAR 83)

HELIOS est un synthétiseur d'images réalistes à degré très élevé d'intéractivité. Il propose un grand nombre de fonctions graphiques :

- Pavage et projections tridimensionnelles de textures colorées.

- Les calculs d'éclairage prennent en compte la brillance des matériaux, la position et la couleur de la source lumineuse ainsi que la couleur ambiante.

- Possibilité d'effacement sélectif.

- Possibilité de désigner un objet sur l'écran à l'aide d'un réticule.

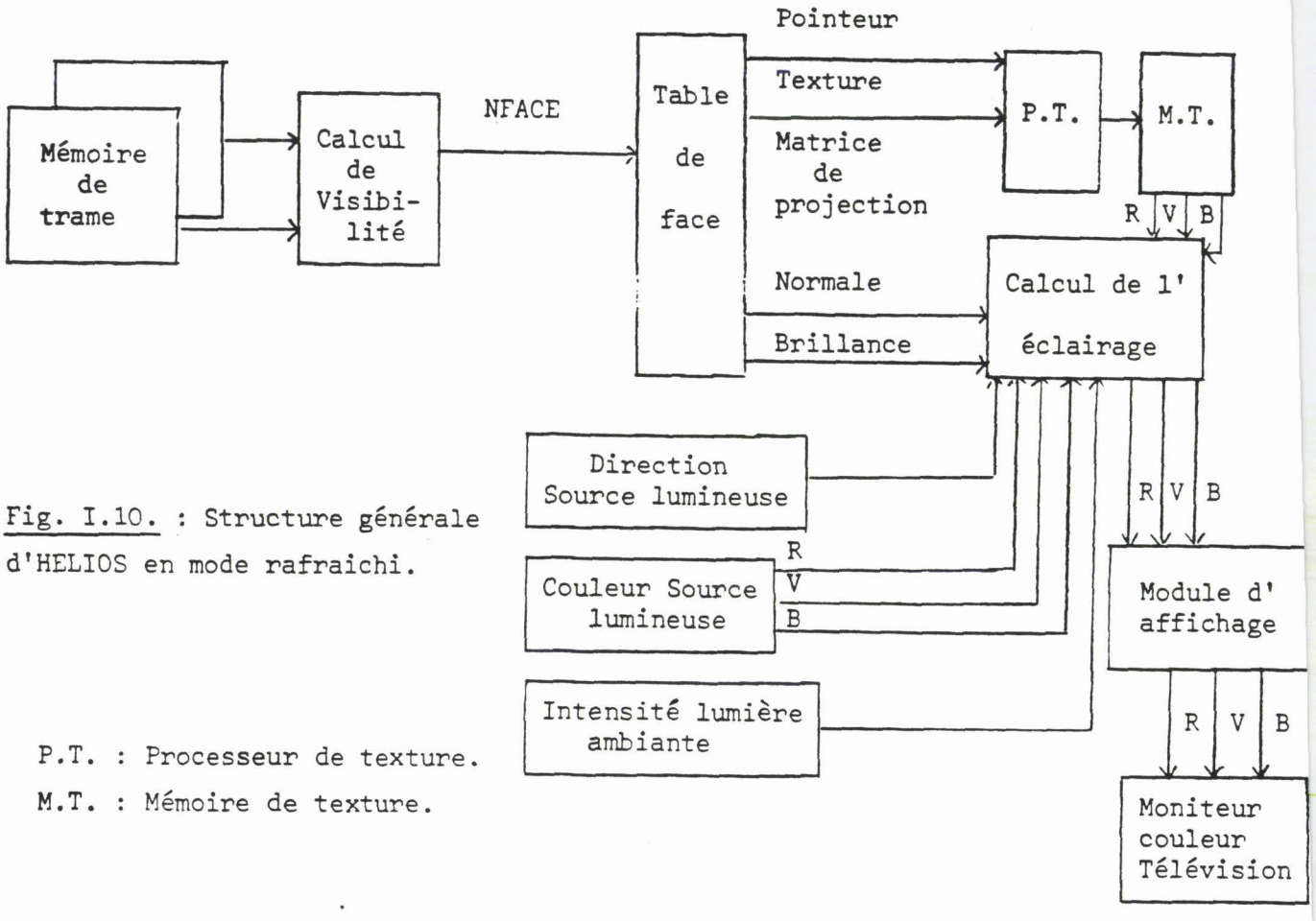


Fig. I.10. : Structure générale d'HELIOS en mode rafraichi.

P.T. : Processeur de texture.

M.T. : Mémoire de texture.

1.4.2. Analyse de quelques logiciels généraux

1- Le logiciel GRIGRI (LLM 78, MAR 80, LUC 77)

GRIGRI est un logiciel graphique intéractif indépendant du contexte d'utilisation. Ce logiciel a été développé au laboratoire d'informatique et de mathématiques appliquées de GRENOBLE.

Ce logiciel ne s'intéresse qu'au dessin dans le plan. Les données à traiter peuvent être de deux types : ensemble de coordonnées ou de textes.

Il est possible de définir un espace de départ (utilisateur) et un espace d'arrivée (écran) ; la transformation faisant passer d'un espace à l'autre étant réalisée automatiquement.

Un dessin est structuré à deux niveaux :

- Un nom de figure permettant de regrouper un ensemble de données.
- Une corrélation permettant de distinguer des éléments à l'intérieur d'une figure.

Un certain nombre de primitives permettent :

- La manipulation de systèmes de coordonnées ; ce sont les primitives FENETRE (utilisateur) et CLOTURE (écran).
- La déclaration de données ; POINTS et TEXTES.
- L'interprétation de données ; primitive MODE.
- L'affichage ; primitive AFFICHER.
- L'effacement sélectif de figures sur l'écran.
- etc...

D'autres primitives permettent de fournir à l'opérateur des indications sur le déroulement d'un programme assurant ainsi l'intéractivité du système.

2- GKS et le core

Ces logiciels sont issus d'une volonté de normalisation de logiciels graphiques. Le core (cf. I.3., 5.1) répond aux recommandations du SIGGRAPH (Special interest group on computer graphics) de l'ACM (Association for Computer Machinery) publié dans (GSP 79).

Le système GKS (Graphical Kernel System) (EEK 79, EEK 80) est issu des travaux de l'organisme de standardisation allemand "DIN". Il constitue l'interface entre le programme d'application et le système graphique. C'est un ensemble d'utilitaires (remplissage, traitement de segments, police de caractères,...). GKS ne permet cependant pas de créer des dessins en 3D. Des extensions ont été proposées pour résoudre ce problème. GKS est devenu la norme ANSI × 33.

3- Les langages évolués

Ce sont des langages graphiques de haut niveau dont la majorité constitue une extension de langages existants ; l'intérêt d'une telle extension étant, à notre avis, double : d'une part les concepteurs n'ont pas à redéfinir un langage de programmation, d'autre part, l'extension se fait le plus souvent sur des langages connus des utilisateurs permettant ainsi une bonne exploitation de ceux-ci.

On trouve dans cette catégorie le langage FORTRAN 3D développé par L'INRIA, ILP (Intermediate Langage for Pictures) conçu par le centre de mathématiques d'AMSTERDAM sur ALGOL 68 (SIN 80, BRA 81).

D'autres langages évolués permettent de par leur structure, leur vitesse d'exécution ainsi que les configurations matérielles sur lesquelles ils fonctionnent d'offrir des mécanismes de synthèse d'images. Signalons à cet effet, et sans entrer dans les détails, le langage LOGO (PAP 71, BOR 83). Le plus puissant et le plus spectaculaire de ces langages est sans nul doute le langage SMALLTALK (COI 82 COI 83).

SMALLTALK est un langage orienté objet développant une programmation fonctionnelle (successeur de LISP) au même titre que PLASMA, FORMES... Les axiomes de base du langage sont au nombre de 4 :

- Toute entité SMALLTALK est un objet, cet axiome vise a unifier la représentation de l'ensemble des données manipulées par le langage. Un objet étant la composante du doublet <état, action>, l'état concerne l'ensemble des connaissances de l'objet (variable rémanentes), l'action étant l'ensemble d'actions que l'objet et capable d'exécuter.

- Tout objet SMALLTALK est instance d'une classe : Une classe regroupe des objets équivalents, notion issue de la théorie des ensemble et que l'on retrouve dans le langage SIMULA.

- Tout objet est activé à la réception d'un message : l'objet est activé lorsqu'il reçoit une demande suivie du nom de l'action qu'on lui demande d'effectuer.

- Toute classe est sous-classe d'une autre classe : ce principe permet de réaliser un mécanisme d'héritage. Dans SMALLTALK 76 le mécanisme d'héritage est simple ; une classe ne peut hériter que d'une sur-classe alors que dans SMALLTALK 80 une classe peut être sous-classe de plusieurs autres (héritage multiple). Cette deuxième approche confère à la structure non plus une arborescence (cas de SMALLTALK 76) mais un graphe ce qui rend complexe l'algorithme de recherche. Le langage se propose une application dans les domaines généraux de l'intelligence artificielle.

I.5. Conclusion

Nous avons essayé, dans ce chapitre, de faire un tour d'horizon sur les différents éléments caractérisant un système graphique notamment dans le cas de la synthèse d'images. Il en ressort surtout que les composants d'un tel système (matériels et logiciels) varient en fonction de l'application concernée. Notons cependant que la différenciation entre les actions effectuées par logiciel et celles effectuées au niveau du matériel n'est pas toujours très nette. En effet actuellement la tendance est à intégrer le maximum de ces fonctions au niveau du matériel, ceci étant dû au faible coût et à la rapidité des composants électroniques.

Le chapitre suivant décrit l'aspect matériel d'un prototype parallèle conçu et réalisé au laboratoire d'informatique de l'université de LILLE 1.

SOMMAIRE DU CHAPITRE II

II. LE SYSTEME PROPOSE

II.1. Introduction (FAK 83, HLSM 82)

De nombreuses applications des systèmes informatiques nécessitent de ces dernièrs de hautes performances en vitesse d'exécution, capacité de stockage et de traitement ainsi qu'en temps de réponse notamment dans le cas d'applications en temps réel. La solution qui semble être la plus immédiate est l'utilisation simultanée de plusieurs éléments de traitement. Une telle démarche a été pensée et mise en oeuvre dès les années 50 (ILLIAC IV, PEPE,...), se posent alors les problèmes et questions suivantes :

- Coûts du matériel et du logiciel prohibitifs.
- Difficulté de mise en oeuvre.
- Optimisation difficile.
- Comment "paralléliser" des programmes séquentiels et les faire partager entre les différents sites ?
- Sur quels critères se baser pour élire un site d'exécution ?
- Comment organiser les synchronisations et les communications entre processeurs ?

Aujourd'hui les progrès extraordinaires de la micro-informatique tendent à résoudre certaines de ces questions, notamment le coût des composants a diminué nettement, d'autre part les circuits intégrés permettent une plus grande facilité de mise en oeuvre et de conception. D'un autre côté on constate une plus grande "maturité" dans la conception de programmes parallèles dûe en grande partie à une expérimentation plus large.

Malgré cela, le problème existe toujours ; car la mise en parallèle de plusieurs unités de traitement ne signifie pas obligatoirement une augmentation des performances du système ; en effet, il se pose d'autres problèmes dûs à la configuration même de celui-ci, le plus important étant, sans doute, celui dû aux conflits d'accès à des ressources communes ; phénomène qui peut ralentir beaucoup les performances du système.

II.2. Les architectures matérielles (FAK 83)

Plusieurs essais sur la classification des architectures de machines ont vu le jour, cependant aucune ne permet jusqu'à présent de fournir tous les paramètres caractérisant ces systèmes. Nous retiendrons cependant les points de distinction suivants :

a - La structure logique qui détermine l'organisation des processeurs ainsi que les relations définissant le contrôle entre-eux. Il existe deux organisations principales :

a-1 - L'organisation verticale (ou hiérarchique) :

Dans cette organisation, il existe un processeur maître et plusieurs esclaves avec les caractéristiques suivantes :

- Les processeurs sont logiquement différents.
- Les communications interprocesseurs doivent transiter par le maître.
- Chaque processeur peut devenir maître à son tour (fixation de priorités).

Dans cette catégorie rentrent naturellement les systèmes multibus (cf. Chapitre III).

a-2 - L'organisation horizontale (ou non hiérarchique) :

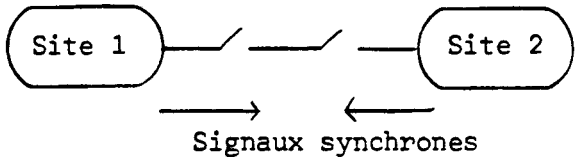
Cette organisation nécessite des moyens de coordination plus sophistiqués. Ses caractéristiques principales sont les suivantes :

- Tous les processeurs sont logiquement équivalents.
- La coordination et le contrôle peuvent être ou non assurés par un processeur maître.
- Chaque élément du système peut communiquer avec n'importe quel autre élément.

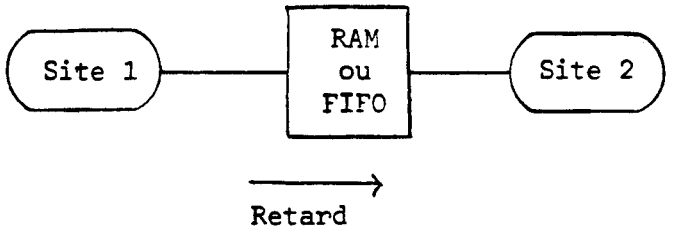
b - La stucture physique qui définit la manière dont l'information transite dans le système et détermine ainsi la configuration de l'outil de communication ainsi que la topologie selon laquelle les éléments sont connectés.

b-1 - L'outil de communication :

L'information transitant entre les processeurs peut être véhiculée soit par l'intermédiaire d'une mémoire commune (on parle alors de commutation de message) ou un bus (on parle alors de commutation de ligne). Dans le premier cas tous les transferts se font par l'intermédiaire d'une mémoire commune et les processeurs n'ont pas d'accès direct les uns vers les autres, dans le second cas par contre tous les processeurs communiquent entre-eux par l'intermédiaire de bus. On parle de structure centralisée quand il s'agit d'un outil de communication unique dans le système (une mémoire commune ou un bus commun). Lorsque l'outil de communication dispose de n voies spécialisées, on parle alors de structure décentralisée.



a - Commutation de ligne.



b - Commutation de messages.

Fig. II.1. : Outils de communication.

b-2 - La topologie d'interconnexion

Nous devons aussi considérer comment sont connectés ensemble les différents éléments, ceci étant très important pour caractériser la fiabilité ainsi que les possibilités d'extension du système. La Fig. II.2 montre les quatre schémas d'interconnexion de base :

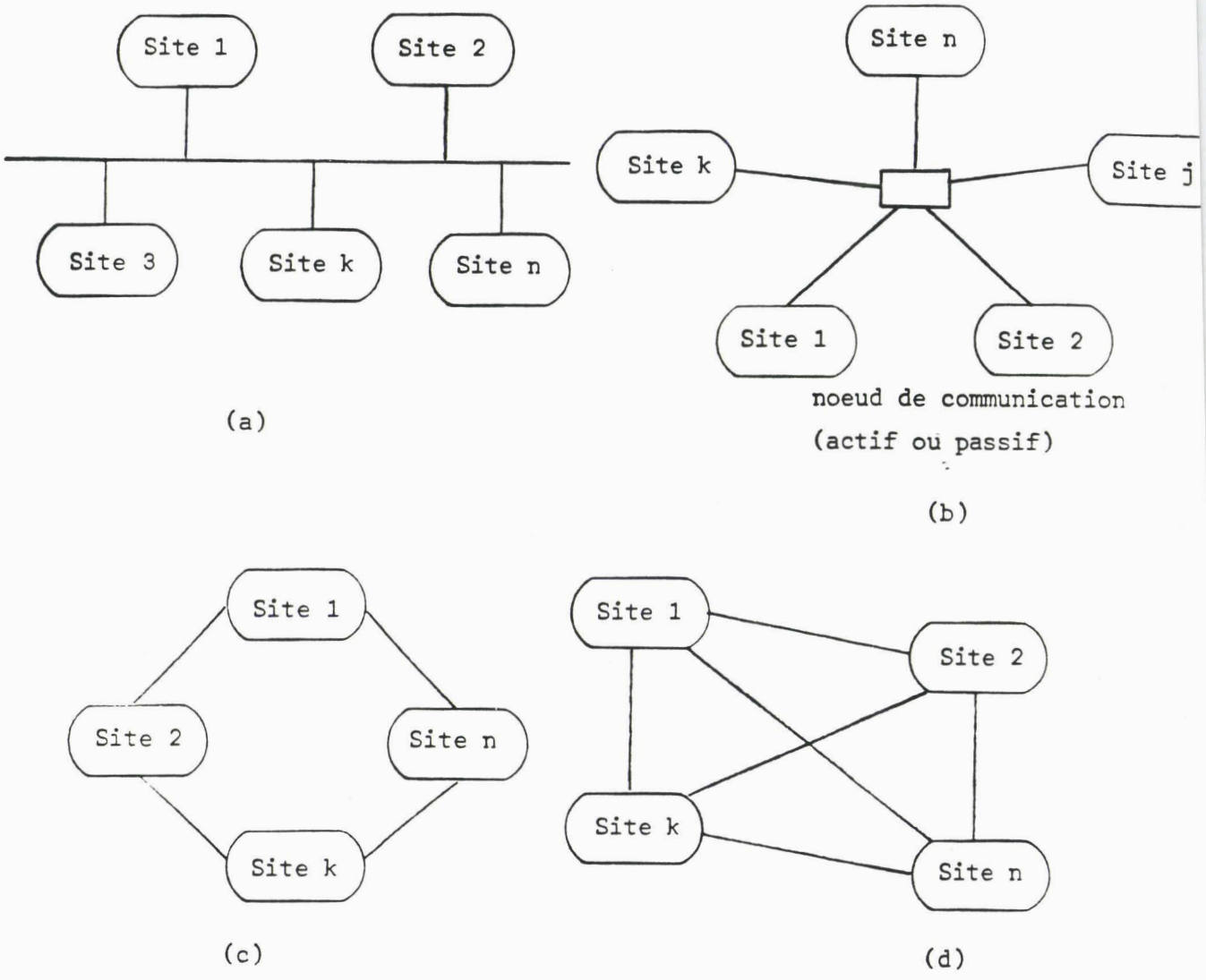


Fig. II.2. : Topologies d'interconnexion.

(a) Partage d'un outil commun d'interconnexion.

(b) Interconnexion en étoile.

(c) Interconnexion en anneau.

(d) Interconnexion complète.

c - Le mode d'intéraction

Ce critère est très important pour la classification d'un système ; il détermine le facteur de couplage ainsi que la nature et le taux de communication entre les différents sites.

Le couplage détermine la fréquence d'échange entre les sites et par cela le taux d'occupation des ressources partagées. Il existe trois formes de couplage le couplage faible, modéré et fort.

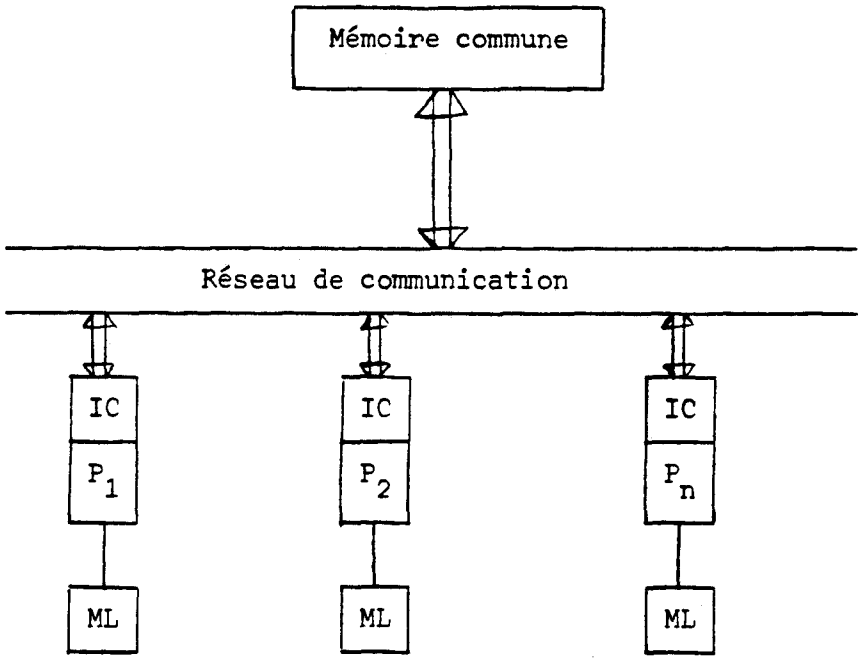
c-1 - Le couplage faible (ou couplage lâche)

Dans ce cas, chaque site dispose de moyens matériels et logiciels qui lui permettent une grande indépendance de traitement, on parle alors de réseaux d'ordinateurs.

c-2 - Le couplage fort

Les sites d'exécution se partagent des ressources indispensables au traitement qu'ils effectuent. On a dans ce cas les caractéristiques suivantes :

- Partage d'une mémoire commune. Le plus souvent, chaque processeur est pourvu en plus d'une mémoire locale (ML) et d'interfaces de communication (IC).



- Un système d'exploitation commun permet la coordination des différents sites.

- Les sites sont le plus souvent homogènes permettant ainsi la banalisation de fonctions de programmation et l'égalité de puissances de traitement.

Le réseau de communication constitue un goulot d'étranglement dans un tel système, c'est en fait lui qui détermine l'ordre de performances d'une telle configuration. Plusieurs études ont été menées mettant en oeuvre de tels réseaux (FRA 81, TSE 81) nous ne rentrerons pas dans ces détails.

De tels systèmes nécessitent des moyens de synchronisation pour l'accès aux ressources communes.

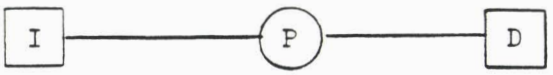
c-3 - Couplage moyen (ou modéré)

Les deux catégories évoquées en c-1 et c-2 correspondent aux cas extrêmes de couplage, le cas intermédiaire correspond à une combinaison de ces deux formes de couplage. On trouvera dans (DEL 79) une étude de tels systèmes.

d - Le mode de fonctionnement

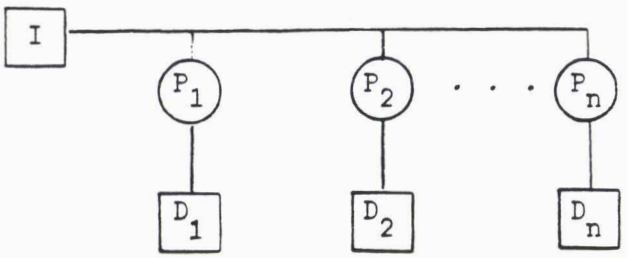
L'une des premières classifications de système informatique revient à FLYNN (FLY 72) où il prend en compte les flux d'instruction et de données. Il distingue quatre grands modes :

- Le mode SISD : Single Instruction, Single Data. Dans ce cas un flux d'instruction (I) se déroule sur un flux de donnée (D) :

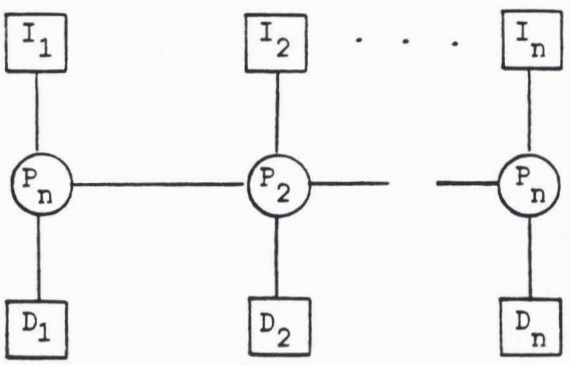


Ce mode correspond principalement au cas de monoprocesseurs.

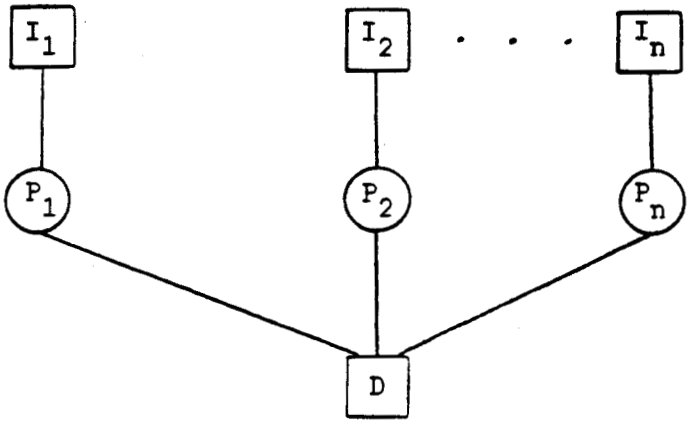
- Le mode SIMD : Single Instruction Stream, Multiple Data Stream. Un flot d'instructions opère sur plusieurs flots de données.



- Le mode MIMD : Multiple Instruction Stream, Multiple Data Stream. Plusieurs flots d'instructions opèrent sur plusieurs flots de données.



- Le mode MISD :



Cette classification ne considère l'exécution qu'au niveau de l'instruction ceci est insuffisant pour caractériser un système. Des modifications à cette classification ont été proposées, l'une d'entre-elles considère l'exécution au niveau des tâches ; une tâche étant un ensemble défini d'instructions (HIG 73, PRE 77).

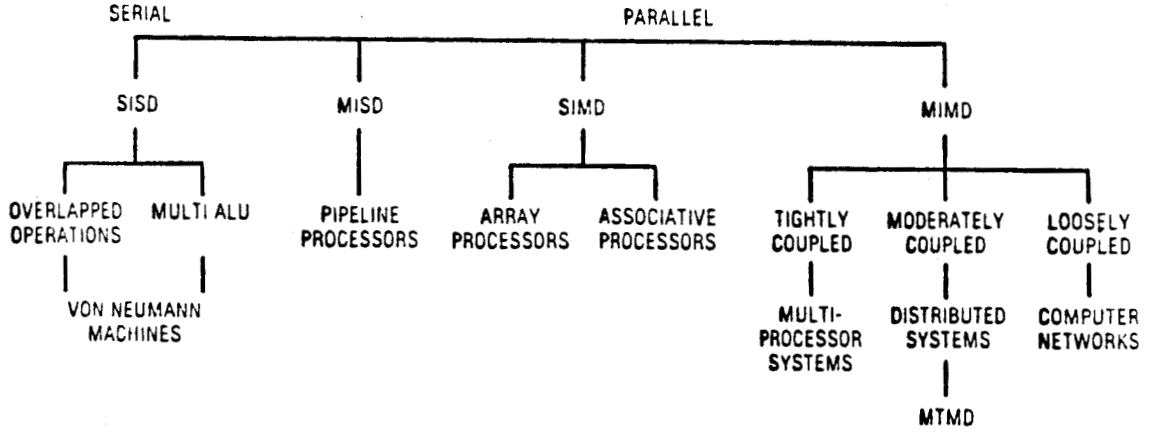


Fig. II.3. Arbre de classification des différentes organisations de processeurs.

e - Conclusion

Malgré les considérations énoncées ci-dessus, il est très difficile sinon impossible de dire que telle ou telle machine appartient à une classe bien déterminée et établir ainsi une classification de toutes les architectures existantes à fortiori mais permet de prévoir les modèles des machines à venir.

II.3. Emploi de systèmes parallèles en synthèse et en animation d'images

La synthèse et l'animation d'images réalistes requièrent des moyens de traitement puissants et rapides (cf. Chapitre I). Les réalisations et études actuelles permettent de dégager deux grandes tendances :

a - Les machines pipe-lines : L'organisation des traitements se fait en cascade. Le processus de production de l'image est décomposé en modules chacun étant confié à un ensemble d'éléments spécialisés de traitement (module d'étude de visibilité, module d'affichage,...). La vitesse de traitement de la chaîne est égale à celle du module le plus lent

Cette structure est de loin la plus répandue mais présente un certain nombre d'inconvénients :

- Difficulté d'extension du système au prix d'un redécoupage du processus.
- Difficulté d'équilibrage des divers composants.
- Problème de stockage réparti.

b - Partition de l'image : L'image est découpée en zones attribuées chacune à un processeur de traitement. Le parallélisme ne se fait plus entre des tâches différentes mais au niveau de chaque tâche.

Le problème est de déterminer le découpage de l'image entre les processeurs assurant ainsi une charge équitable entre les différents processeurs et définissant ainsi leur nombre et leurs performances.

Le mode de partition le plus simple consiste à décomposer l'image en zones contigües de tailles fixes et égales (SCH 80).

La suite de ce chapitre décrit un système fonctionnant selon ce principe.

II.4. Le système proposé

Le système étudié vise à expérimenter des algorithmes parallèles de synthèse, de manipulation et d'animation d'images. Ce sont là des domaines nécessitant de grandes performances en temps d'exécution ainsi qu'en capacités de traitement et de stockage.

Pour l'animation en temps réel, l'élaboration d'une nouvelle image doit se faire toutes les 40 ms, si cette image est destinée à être affichée sur un écran de télévision (40 ms correspondent au temps d'une trame de télévision).

Si chaque pixel de l'image à afficher est codé sur q bits, la mémoire de trame destinée à rafraîchir un écran de télévision doit avoir une capacité de $\ell \times p \times q$ bits où $\ell \times p$ est le nombre de points de l'image. Nous voyons alors que la taille de cette mémoire devient très vite importante.

Pour notre application, la taille de l'image est de 256×256 pixels. Chaque pixel est codé sur 8 bits, offrant ainsi une gamme de 256 couleurs. Ceci confère à la mémoire de trame une capacité de 64 K octets. Pour l'obtention d'une image carrée sur l'écran, celle-ci est affichée toutes les demi-trames, soit toutes les 20 ms. Le temps moyen de traitement d'un pixel est d'environ $20/64.10^3 \simeq 300$ ns. Si l'on suppose que le traitement d'un pixel correspond à n instructions machine, chacune d'entre-elles doit avoir une durée moyenne de (300/n)ns. Les technologies actuelles des microprocesseurs et des mémoires ne permettent pas d'assurer des temps d'exécution pour $n > 2$.

Ceci nous a conduit à considérer un partitionnement de la mémoire d'images entre plusieurs processeurs ; chacun d'entre-eux accède à une tranche de celle-ci. Dans ce cas si K est le nombre de ces processeurs, K pixels peuvent être traités simultanément.

L'architecture globale est déterminée par différents critères :

- Coût du processeur élémentaire et des circuits associés.
- Vitesse d'exécution.
- Nombre de processeurs.
- Faisabilité et simplicité de mise en oeuvre.

La microprogrammation apporte plusieurs solutions et avantages :

- Grande souplesse de mise en oeuvre (voire adaptabilité).

- Grande efficacité ; les micro-instructions sont plus performantes que les instructions car elles permettent le contrôle des éléments internes du mirco-processeur.

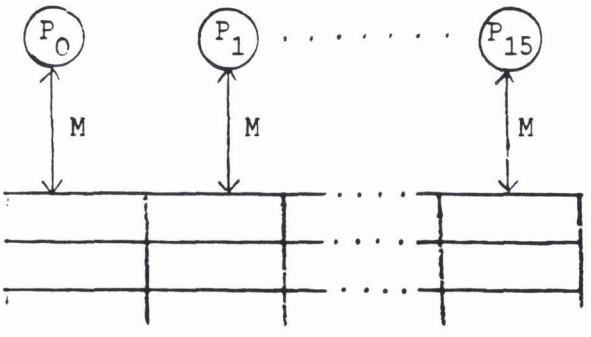
L'implémentation de microprogrammes ne présente pas un caractère figé et définitif. Il est possible de les modifier en vue de l'évolution des fonctions implémentées.

Les microprocesseurs microprogrammés actuels peuvent exécuter des micro-instructions en moins de 200 ns (AMD 2900...). Pour notre part, nous avons choisi le microprocesseur microprogrammable SIGNETICS 8 × 300 dont nous donnerons des détails dans la suite de ce chapitre. Ce choix est surtout dû à la simplicité de mise en oeuvre de ce circuit et à la vitesse d'exécution (250 ns pour une micro-instruction quelconque).

Ceci nous a amené à réaliser un réseau de 16 processeurs parallèles accédant chacun à une tranche de 4 K octets. Par conséquent 16 pixels peuvent être traités simultanément. Le temps moyen de traitement de 16 pixels devient alors 16 × 300 = 5 ms. Soit 20 micro-instructions par pixel.

II.4.1. Organisation du traitement

Les 16 processeurs de traitement accèdent en mémoire d'images chacun par une fenêtre de 8 bits (un pixel). 16 pixels sont traités en parallèle.



P_i = Processeur de traitement

M = fenêtre d'accès en mémoir d'images.

Fig. II.4. : Accès en mémoire d'images.

La mémoire d'images est décomposée en plans (4 dans la configuration actuelle). Un plan mémoire correspond à la taille de l'écran. Un dispositif cablé ; le convertisseur permet l'accès à un plan de celle-ci et sa visualisation sur un visuel (Dans notre cas le convertisseur entretient l'image affichée sur une télévision couleur). Nous verrons dans le Chapitre III plus en détail le rôle et l'importance du convertisseur dans la synthèse et l'animation d'images. Ceci justifie les intéractions entre le convertisseur et le reste du système (Fig. II.5.).

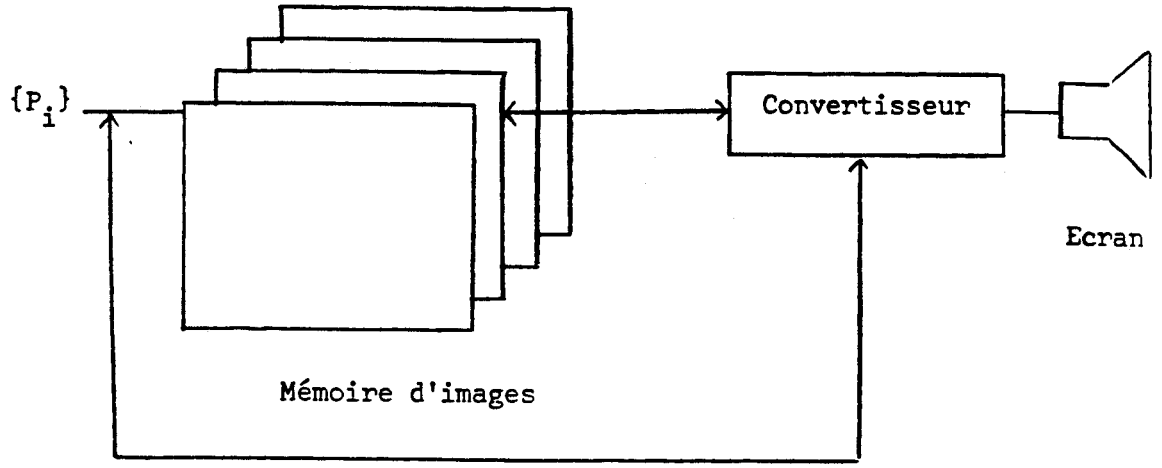


Fig. II.5. : Moyens de synthèse et d'entretien de l'image.

Le format de l'adresse en mémoire d'images correspond ainsi au triplet <ℓ, c, z> où ℓ désigne un numéro de ligne ($0 \leq \ell \leq 255$), c un numéro de colonne ($0 \leq c \leq 15$) et z un numéro de zone ($0 \leq z \leq 3$).

Cette adresse est donc codée sur 8 + 4 + 2 = 14 bits. Chaque processeur exploite 1/16 de cette mémoire selon le schéma de la figure II.6..

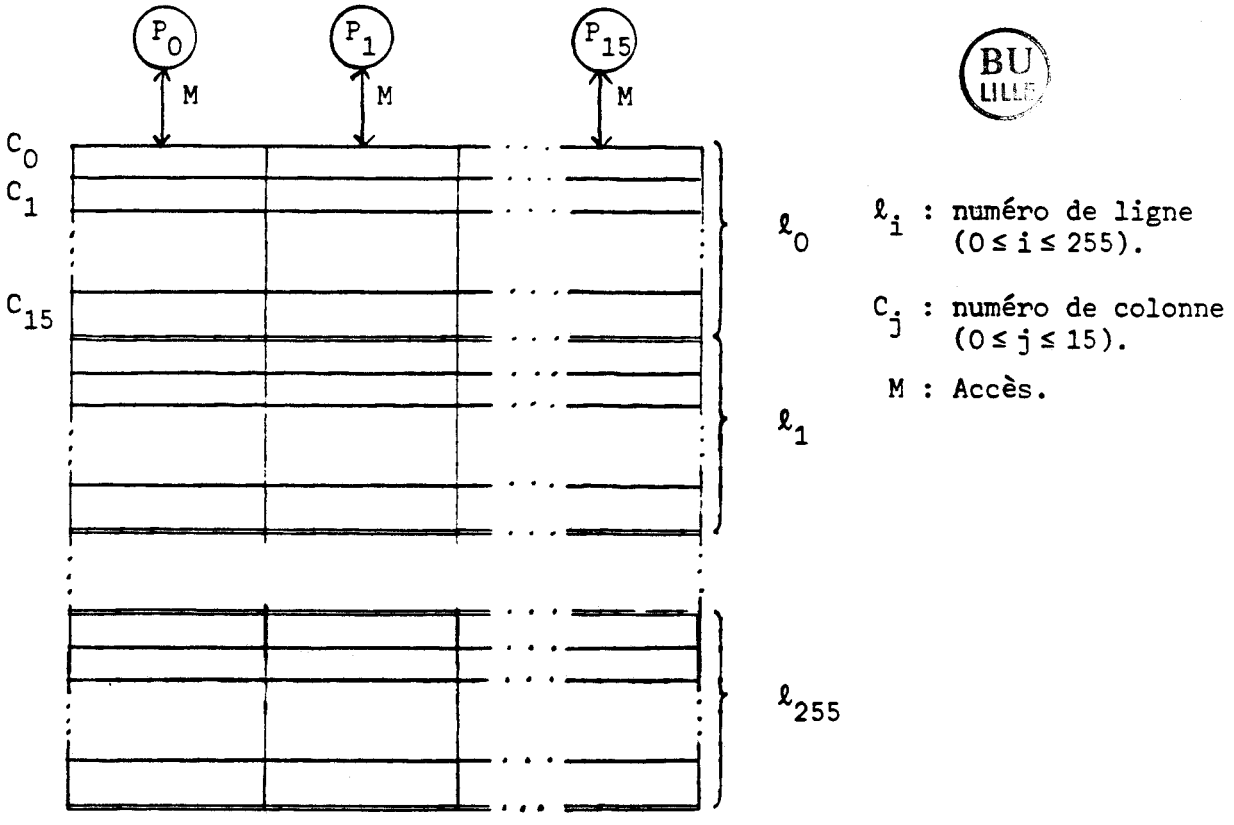
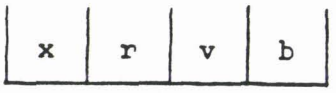


Fig. II.6. : Organisation de la mémoire d'images.

Un pixel est codé de la façon suivante :



b : Codage du bleu
v : Codage du vert
r : Codage du rouge
x : Informations servant à la définition du pixel (profondeur,...)

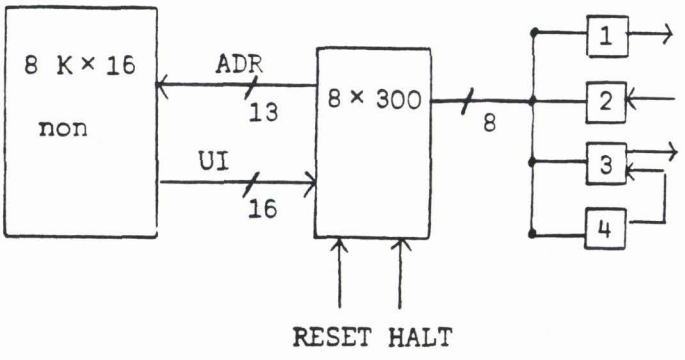
II.4.2. Parallélisme du traitement

Les processeurs doivent concourir à l'exécution de tâches parallèles et coopérantes exploitant la mémoire d'images. Cette initiative requièrt que soient prise en compte des contraintes temporelles définissant les notions de séquencement des processus se déroulant sur les différents sites ainsi que les notions de synchronisation entre ces différents processus, en plus des domaines spatiaux doivent être mis en oeuvre permettant les communications et les routages entre les processeurs.

Les contraintes temporelles de séquencement sont assurées par un processeur maître ainsi qu'une horloge commune.

II.4.3. Le processeur élémentaire

Les nécessités de fonctionnement évoquées précédemment ont conduit au choix du microprocesseur micro-programmable SIGNETICS 8 × 300 (SIG 79) dont certaines caractéristiques sont présentées figure II.7.



(1) : Sortie.
(2) : Entrée.
(3) : Bidirectionnel.
(4) : Sortie (2 bits sélectionnent le sens de (3).
RESET : Initialisation.
HALT : Arrêt.

Ce microprocesseur exécute en 250 ns une micro-instruction codée sur 16 bits issue d'une mémoire de contrôle de 8 K mots. La micro-instruction est composée de la façon suivante :

- 3 bits (poids forts) : code opération.
- 1 à 4 champs de micro-commandes pour représenter des constantes, des registres internes ainsi que le contrôle de rotation ou masques sur les données.

Le 8 × 300 peut reconnaître 8 micro-instructions, ceci forme un répertoire assez réduit mais pouvant être très étendu par le biais de la micro-programmation : Ces instructions sont des instructions arithmétiques et logiques (addition, et, ou exécutif), de transfert, de rupture de séquence (saut conditionnel, inconditionnel, branchement avec retour éventuel). Ces instructions s'exécutent entre une source et une destination pouvant être des registres internes ou externes au microprocesseur, des constantes ou des adresses.

Le microprocesseur est piloté par une horloge externe.

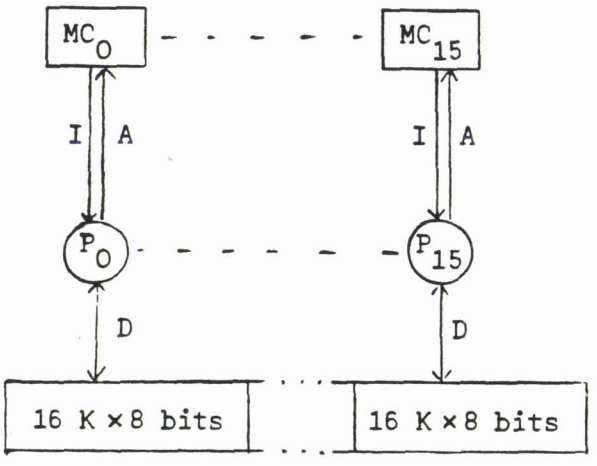
La rapidité, la souplesse et la simplicité de configuration sont des critères essentiels pour le choix d'une telle famille.

II.4.4. Choix de l'architecture

L'architecture la plus classique est sans doute celle où le réseau de processeurs accède à une mémoire commune correspondant ainsi à un fort couplage entre les processeurs.

Cependant l'examen d'une telle architecture révèle son extrême difficulté de mise en oeuvre et sa complexité dûes au réseau de commutation servant l'interface entre les processeurs et la mémoire. Les performances d'un tel système sont fonction du réseau. Des études approfondies sur la question (TSE 81) montrent qu'il s'ensuit des temps d'overhead incompatibles avec le temps réel.

Cette solution a donc été écartée au profit d'une autre plus simple et plus économique compte tenu de l'évolution des circuits impliquant une baisse du coût de ceux-ci. Cette deuxième approche confère à chaque processeur sa mémoire de commande locale ; il n'exécute d'instructions qu'en provenance de celle-ci.



Le degré d'intéraction entre les processeurs n'est pas limité par cette architecture à condition d'accepter que les mêmes modules de programmes soient répliqués dans les différentes mémoires de commandes. Les performances du système en MIPS sont quasiment égales à la somme des performances en MIPS des différents sites (Un site est alors l'ensemble $\{MC_i, P_i, M\}$) soit ici 64 MIPS.

De plus, et moyennant l'existence d'une circuiterie complémentaire que nous définirons par la suite, une telle architecture est apte à différents modes de fonctionnement :

- Mode MIMD : Chaque processeur étant autonome grâce à sa mémoire locale, ce fonctionnement est rendu possible.

- Mode SIMD : Il est rendu possible en dupliquant les même séquences de microprogrammes dans les différentes mémoires locales. Un contrôle extérieur doit déterminer les cohérences temporelles en agissant par exemple sur les horloges relatives aux différents sites. Ceci implique l'existence d'outils rapides de synchronisation.

En outre les processeurs doivent coopérer à l'exécution de tâches concurrentes et doivent ainsi s'échanger des informations.

II.4.5. Conclusion

Ces observations nous ont conduit à définir la structure suivante :

- Existence d'un réseau d'échange et de communication qui doit être rapide et bidirectionnel, son rôle est d'assurer la coopération entre les différents processeurs.

- Un réseau de synchronisation rapide et simple à mettre en oeuvre permet de fixer les cohérences temporelles entre les processeurs.

- Nécessité d'un superviseur dont le rôle fondamental est d'assurer le fonctionnement intéractif et collectif des différents processeurs de traitement.

- sites de même nature, organisations matérielles identiques.

- Pour assurer un fonctionnement cohérent, une même horloge pilote l'ensemble du système. Soit une architecture a organisation horizontale.

II.5. Le processeur maître

L'organisation du contrôle ainsi que la supervision des traitements se déroulant sur les différents sites sont confiés à un processeur particulier ; le processeur maître. Ce processeur s'occupe aussi d'assurer les échanges avec l'extérieur. Les fonctions exécutées par le processeur maître sont complémentaires de celles exécutées par les processeurs de traitement ; en effet, alors que ces derniers exécutent des séquences de microprogrammes manipulant les images stockées en mémoire, le processeur maître s'occupe du contrôle de ces séquences ainsi que de l'acquisition de nouvelles consignes émises par les programmes de l'utilisateur afin d'aiguiller les traitements vers les microprogrammes concernés.

Nous distinguons ainsi deux niveaux différents de commande : le processeur maître exécute des commandes de haut niveau, les processeurs de traitement exécutent des commandes de bas niveau.

Le rôle du processeur maître est donc multiple :

- Initialisation et contrôle de séquences microprogrammées se déroulant sur les différents sites.

- Enchaînement de ces séquences.

- Acquisition des données depuis le processeur hôte.

- Acquisition de commandes issues de l'extérieur.

- Diffusion de commandes vers les processeurs du réseau.

- Acquisition de l'état des différents sites.

- Gestion de la mémoire d'images dans certains cas de fonctionnement (adresse, contrôle).

- Gestion des conflits internes à la machine.

- etc...

Les réseaux de communication et de synchronisation s'étendent à ce processeur.

Le processeur maître et conçu à partir des mêmes circuits que les processeurs élémentaires.

La simplicité de configuration et l'homogénéité du matériel et du logiciel sont là aussi des critères considérables qui ont motivés ce choix.

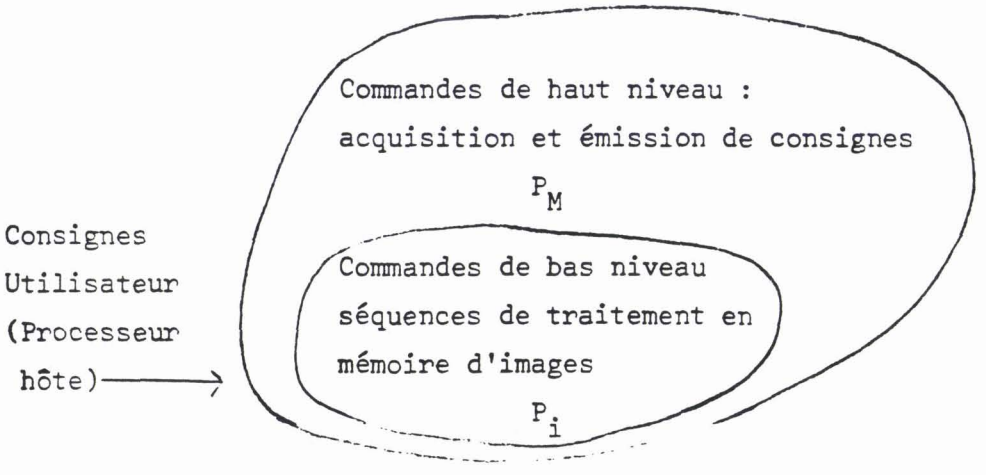


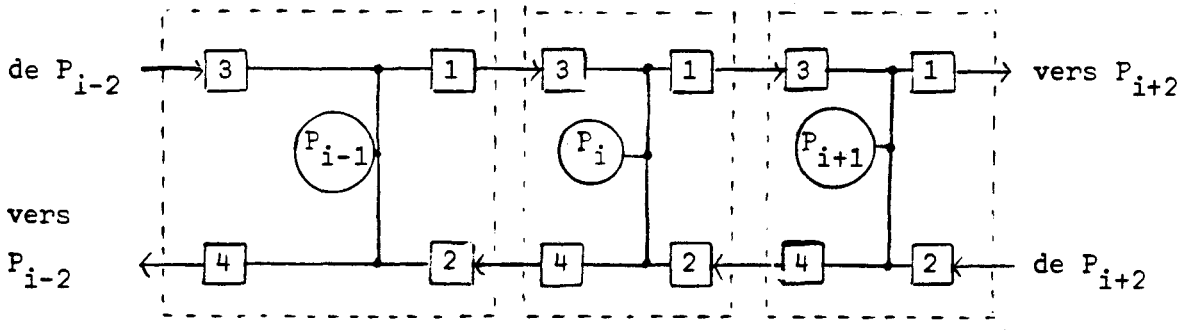
Fig. II.7. : Niveaux de commande.

II.6. Les communications

La coopération entre les différents processeurs exige des outils rapides de communication. Deux processeurs P_i et P_j peuvent communiquer s'ils disposent de ressources partagées ; celles-ci pouvant être une mémoire, un registre ou une ligne. Ainsi plusieurs voies de communications ont été prévues pour les besoins du fonctionnement.

II.6.1. Routage d'une information entre processeurs voisins

Chaque processeur dispose de quatre registres d'interface assurant le routage d'une information (8 bits) entre ce site et les sites voisins "gauche" et "droite".



i modulo 16

1 : Sortie droite (Rout : Right out)
2 : Entrée droite (Rin : Right in)
3 : Entrée gauche (Lin : Left in)
4 : Sortie gauche (Lout : Left out)

De la même façon, ces voies se referment en anneau par l'intermédiaire de P_{15} et P_0.

Par le biais de ces chemins, on peut envisager divers types de fonctionnement contrôlés par microprogrammes.

a-Fonctionnement pipe-line

Dans un fonctionnement pipe-line un processus est décomposé en plusieurs sous-processus, chacun d'entre-eux est confié à une unité d'exécution autonome et spécialisée (COR 78). On peut supposer qu'une information à traiter transite séquentiellement par différents sites chacun d'entre-eux effectuant un certain traitement selon le schéma suivant :

	P_1	P_2	$\cdots$	P_n
t_1	$X_{1,1}$	-	$\cdots$	-
t_2	$X_{1,2}$	$X_{2,1}$	$\cdots$	-
$\vdots$	$\vdots$	$\vdots$		$\vdots$
t_n	$X_{1,n}$	$X_{2,n-1}$	$\cdots$	$X_{n,1}$

- : attente

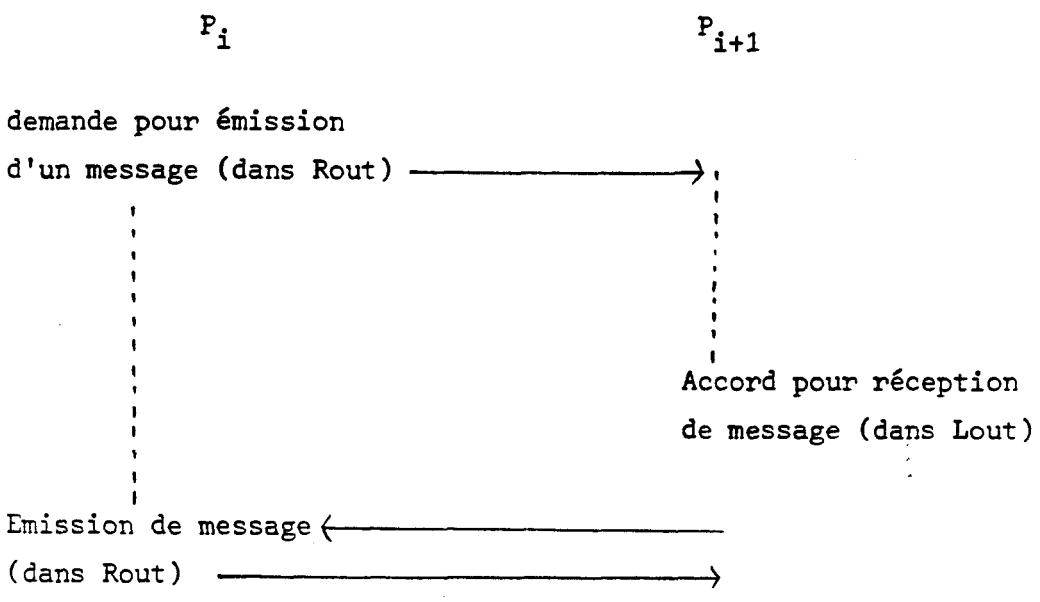
$X_{i,j}$: Exécution par P_i de la séquence j sur les informations fournies à son entré

Dans un tel processus, chaque site reçoit un ensemble de données sur lesquelles il exécute un traitement spécifique et fournit ces données modifiées en sortie.

L'intérêt d'un tel modèle de fonctionnement n'est plus à démontrer, signalons l'existence de plusieurs systèmes logiciels et matériels fonctionnant selon ce principe. (COR 78).

b-Fonctionnement de manière asynchrone

Les outils de routage permettent aussi de prévoir un fonctionnement asynchrone de type FULL-DUPLEX selon un protocole établi par micro-programmes, dans le cas de deux processeurs P_i et P_{i+1}, l'un est producteur et l'autre consommateur.



Cependant une telle démarche risque d'encombrer le réseau de routage et il en résulte un overhead important. La machine utilisera de façon plus rentable des outils cablés de synchronisation.

II.6.2. Voies de communication avec le processeur maître

Le processeur maître doit pouvoir dialoguer avec n'importe quel site du réseau, des voies de communication doivent exister à cet effet. La communication devant être bidirectionnelle.

II.6.2.1. Cas où P_M est émetteur

C'est le cas où une information détenue par P_M doit être diffusée à un ou plusieurs sites du réseau. Cette voie pouvant être exploitée par micro-programmes dans des cas divers concernant par exemple l'envoi d'adresse de début de séquence, de paramètres concernés par un ou plusieurs traitements, d'octets de contrôle... Ces informations peuvent être précédées par un code indiquant le ou les sites concernés. La figure II.8 montre le schéma fonctionnel de la réalisation logique.

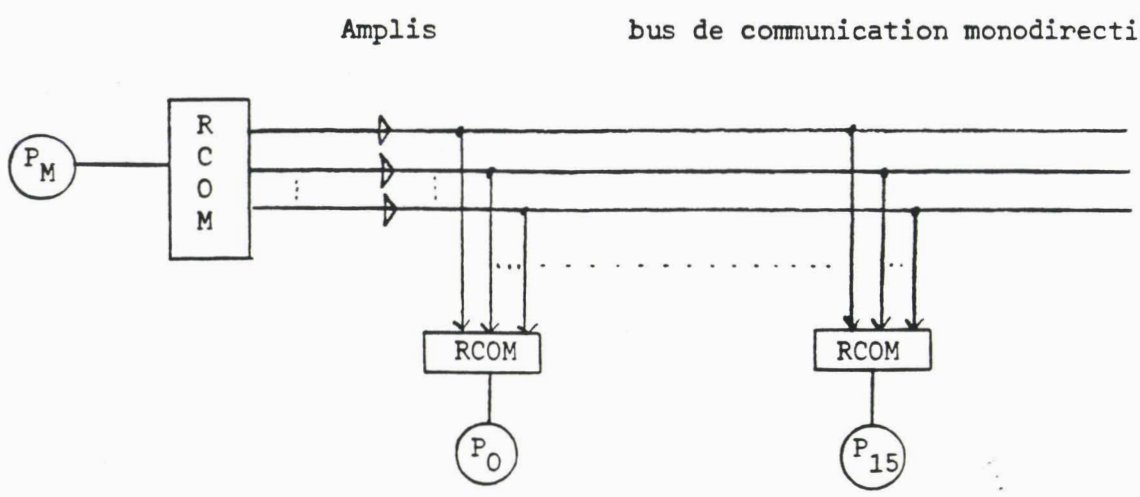


Fig. II.8. : Diffusion 1 → n.

La sélection éventuelle peut se faire également grâce aux éléments spécifiques figurant dans le microprogramme de chaque processeur.

II.6.2.2. Cas où P_M est récepteur

Une deuxième voie permet la liaison dans le sens inverse assurant ainsi la communication n → 1. L'architecture à bus unique nécessite que des précautions de conception soient prises en compte afin d'éviter les conflits d'accès à ce bus. Il doit exister une sélection temporelle du site d'émission d'où un multiplexage n → 1.

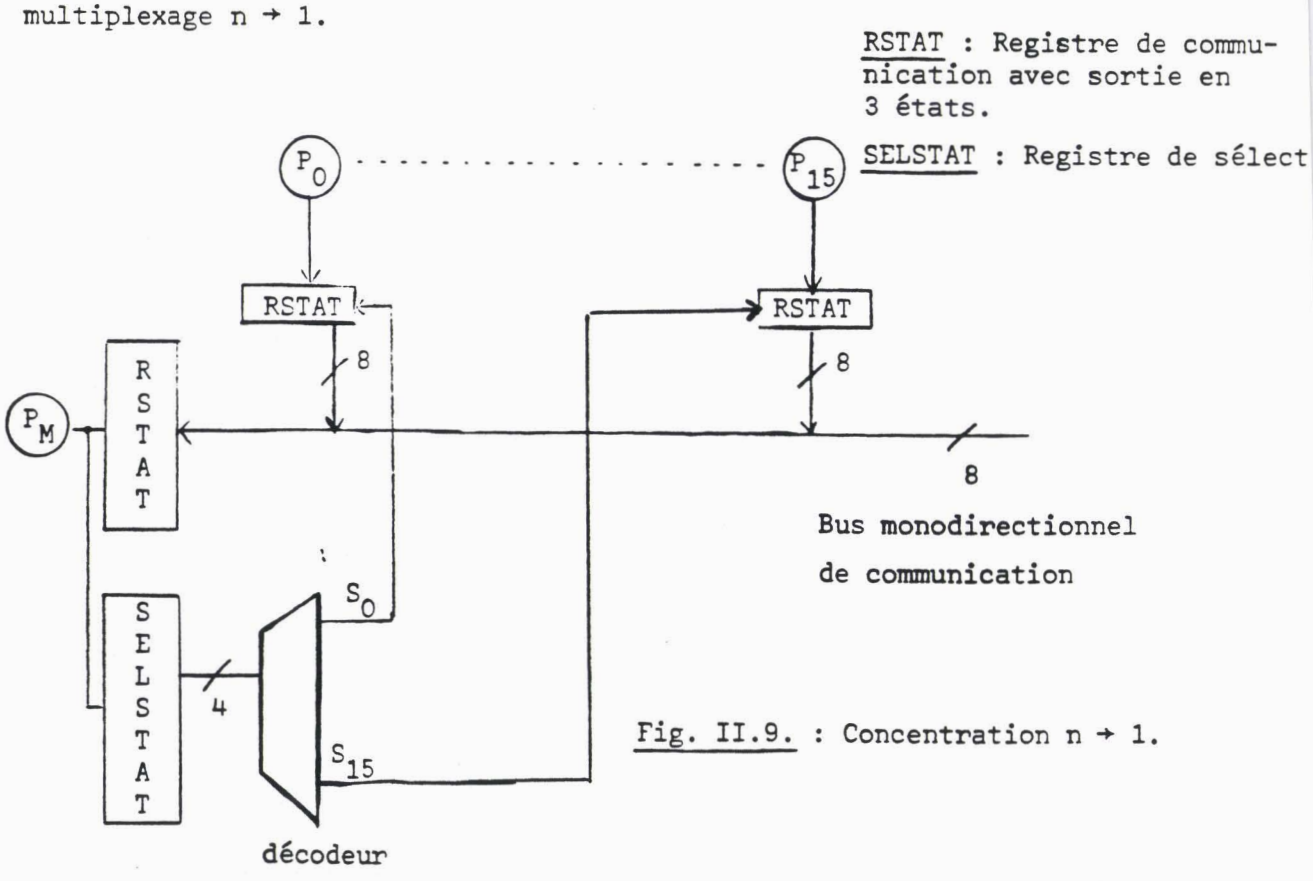


Fig. II.9. : Concentration n → 1.

Le processeur maître affiche le numéro du processeur émetteur dans SELSTAT validant ainsi le registre RSTAT correspondant et inhibant tous les autres. Cette voie permet de véhiculer des informations concernant les états des différents processeurs. Elle peut permettre également à un processeur quelconque d'échanger des informations avec l'extérieur par le biais de P_M.

II.7. Les outils matériels de synchronisation

La synchronisation de processus parallèles revêt des formes diverses et des significations différentes. Elle est indispensable lorsque ces sites doivent concourir à l'exécution de tâches coopérantes. Plusieurs notions différentes sont attachées à cette fonction, en effet des processus peuvent être amenés à devoir se synchroniser pour accéder à une ressource critique ; dans ce cas les processus concernés doivent se synchroniser afin d'accéder en exclusion mutuelle à cette ressource (CRO 76). L'échange d'informations entre processus peut aussi se faire de manière synchrone et éviter ainsi que le consommateur consomme avant production et l'effet inverse. De tels mécanismes ont été initialement étudiés pour leur application dans les systèmes d'exploitation notamment. Plusieurs méthodes ont été adoptées, ainsi la synchronisation par sémaphores (CRO 76), par moniteurs (HOA 74)...

La synchronisation peut aussi servir à signaler l'arrivée d'un processus en un point donné de son évolution et permettre ainsi une quantification temporelle de cette évolution par exemple, ou qu'à ce niveau soit déclenché un processus parallèle.

D'une façon générale, on peut dire qu'il existe un mécanisme de synchronisation entre deux processus si l'un d'eux est mis en mesure de savoir que l'autre occupe ou a probablement occupé un état défini (COR 83).

Lorsque le processus observateur peut obtenir l'information sur l'état de l'autre au bout d'un temps quelconque, on se trouve dans une situation de *synchronisation lâche*. C'est le cas de synchronisation le plus classique. Ainsi un cas de producteur-consommateur nécessite que le consommateur sache que le producteur ait produit, le producteur que le consommateur ait consommé.

La synchronisation est dite *stricte* lorsque le processus observateur doit avoir état du processus observé dans un délai maximum. Ceci intervient surtout lorsque l'information risque de devenir caduque après ce délai. Ce cas intervient surtout dans les fonctionnements de processus en temps réels tels que le contrôle de processus industriels.

La *synchronisation absolue* exige que ce délai soit nul ceci a pour conséquence que les processus soient en mesure d'exécuter en même temps l'action qui suit les primitives de synchronisation.

Dans les systèmes multiprocesseurs, les sites doivent coopérer à l'exécution de tâches parallèles et coordonner leurs activités à des échelles très fines soit à celle de l'instruction ou même de la micro-instruction. Ces contraintes font qu'il doit exister un mécanisme de synchronisation absolue.

Notre travail se situe dans ce cadre en posant comme contraintes :

- Les primitives de synchronisation se présentent à l'échelle des micro-instructions.

- Le nombre de processeurs concerné par une action de synchronisation peut être quelconque.

- Il n'existe aucune hiérarchie entre les processeurs.

II.7.1. Les techniques de rendez-vous

Les conditions de fonctionnement des sites ont été présentés précédemment (LAM).

- Ils sont identiques : mêmes composants, mêmes organisations matérielles.

- Une même horloge pilote l'ensemble des sites.

- Toutes les micro-instructions qu'ils exécutent sont de même durée.

Un *rendez-vous* est un point de passage obligé pour les processus concernés par une action de synchronisation.

Un processus ne peut quitter un rendez-vous une fois qu'il y est parvenu qu'à la condition que tous les autres lui aient signalé y être également parvenus.

En conséquence, les primitives exécutées par les sites après avoir quitté les rendez-vous sont synchrones au sens strict.

Matériellement, la réalisation d'un rendez-vous peut se faire comme suit :

- Chaque P_i dispose d'un indicateur S_i d'arrivée au rendez-vous, et d'un indicateur C d'aquittement.

- A l'initialisation S_i = FAUX.

- P_i arrive au rendez-vous et met S_i := VRAI.

La réalisation du rendez-vous se fait lorsque

$$C = \bigcap_{i=0}^{n} S_i = \text{VRAI}.$$

- Chaque processeur qui arrive au rendez-vous exécute la primitive :

ATTENDRE : tant que C = FAUX aller à ATTENDRE ;

Ce test est fait en une seule micro-instruction (NZT). Ce mécanisme est cablé pour éviter un temps d'overhead dû aux partages de ressources en mode programmé.

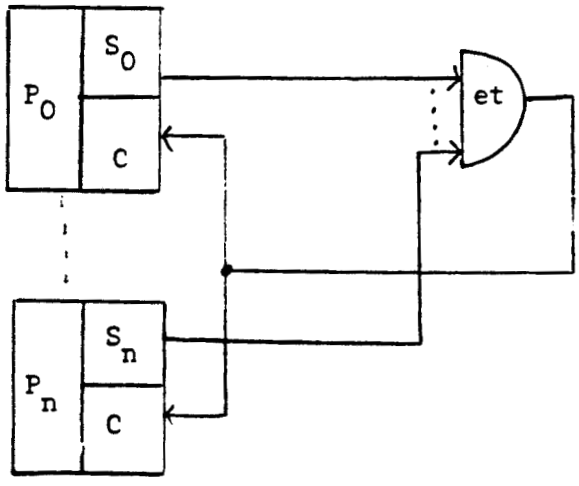


Fig. II.10. : Illustration du principe de rendez-vous.

Cette technique implique quelques contraintes :

- Chaque processus est obligé de passer par le point de rendez-vous sous peine de bloquer tous les autres.

- Les premiers processus arrivés au point de rendez-vous se bloqueront dans l'attente du dernier processus concerné. En fin de compte l'efficacité du système dépend beaucoup des "écarts" entre les arrivées au point de rendez-vous.

- Le fonctionnement défectueux d'un processus peut mener l'ensemble du système dans une situation d'interblocage.

Nous allons détailler dans ce qui suit les mécanismes étudiés et mis en oeuvre exploitant la technique de rendez-vous.

II.7.2. La synchronisation par groupes

Ce mode peut être qualifié de rendez-vous multiples. En effet, on peut imaginer divers découpages d'un réseau de M processus et les unir chacun par une fonction de synchronisation, théoriquement le nombre de groupes pouvant être ainsi formé est : 2^n.

Soit dans le cas de 16 processus : $2^{16} = 2^{10} \cdot 2^6 = 64$ K groupes.

Ce mode répond aux exigences de synchronisation les plus fréquentes du système ; synchronisation entre deux processeurs voisins (pour le routage d'une information par exemple) ; entre l'ensemble des processeurs, etc...

Partant des exigences les plus importantes et les plus fréquentes de la machine et dans le but d'éviter un surchargement du circuit logique de synchronisation, la configuration actuelle autorise les 6 groupes de synchronisation suivants :

G1 : (0-1), (2-3), (4-5), (6-7), (8-9), (10-11), (12-13), (14-15)

G2 : (1-2), (3-4), (5-6), (7-8), (9-10), (11-12), (13-14), (15-0)

G3 : (0-1-2-3), (4-5-6-7), (8-9-10-11), (12-13-14-15)

G4 : (0-1-2-3-4-5-6-7), (8-9-10-11-12-13-14-15)

G5 : Les 16 processeurs du réseau

G6 : 16 processeurs avec le maître

G0 et G7 servent à l'initialisation.

Matériellement chaque processeur dispose d'un registre de synchronisation SYNOUT (3 bits) dans lequel il émet le numéro du groupe auquel il veut participer. Une deuxième couche logique permet de lui adresser le signal de synchronisation dans un registre SYNIN (1 bit).

L'algorithme d'une demande de synchronisation est le suivant :

$\underline{P_i}$

SYNOUT ⟵ Numéro de groupe
WAIT : tant que SYNIN ≠ 0 aller à WAIT
SYNOUT ← 0 (désélection du mode).

II.7.3. La synchronisation par nom

La synchronisation par groupe concerne des groupes de processeurs prédéfinis et figés et entraîne les conséquences suivantes :

- Impossibilité de la formation dynamique de groupes. Ce cas est indispensable dans le cas d'une synchronisation conditionnelle. Soit par exemple un processus P_i qui doit se synchroniser avec un processus P_k si un indicateur B est "vrai" sinon avec un autre processus P_i si B est "faux".

- Ce mode ne recouvre pas tous les cas possibles. Imaginons par exemple que les besoins de fonctionnement font que P_1 doive se synchroniser avec P_{15}, dans le cas actuel et selon les choix qu'on a fait il faux imposer à la machine une synchronisation totale G5, ce qui pénalise beaucoup les performances de celle-ci.

- En outre un processus impliqué dans un groupe est obligé d'y participer et ne peut s'abstenir d'une action de synchronisation.

Pour pallier à ces inconvénients nous avons mis au point une méthode de synchronisation dite synchronisation par nom ou synchronisation associative se proposant donc un fonctionnement plus général selon les mêmes principes définis précédemment.

Les règles de ce second mode sont les suivantes :

- Il existe p noms de synchronisation :

$$S = \{S_0, S_1, \ldots S_{p-1}\}$$

- Le nom S_0 sert de valeur d'initialisation , le processus qui l'affiche indique qu'il n'a pas effectué son choix et qu'il est, de ce fait, indécis.

- Un ou plusieurs processus peuvent s'abstenir d'une action de synchronisation en ne choisissant aucun groupe, dans ce cas ils affichent le nom S_1.

- Tant qu'il existe un nom S_0 affiché, les groupes de synchronisation ne sont pas fixés et aucune synchronisation n'est possible.

II.7.4. La synchronisation par retrait

L'inconvénient majeur de la synchronisation par nom réside dans le fait qu'un processus indécis bloque l'ensemble des groupes y compris ceux qu'il ne peut rejoindre. Ceci nous a conduit à affiner cette notion et à adopter une autre solution dite "par retrait" :

- Dans ce cas ; au lieu d'afficher un nom de synchronisation, le processus affiche tous les noms.

- Chaque nom est représenté par un indicateur booléen N_{ij} correspondant au nom j du processeur i. $N_{ij} = 1$ indique que le processeur i maintient l'hypothèse qu'il peut choisir le nom j.

- Un processeur n'ayant pas encore décidé de son groupe affiche au moins deux noms ceux sur lesquels il est indécis.

- Lorsqu'un processeur choisit définitivement son groupe il n'affiche que le nom de celui-ci.

- Un processeur désirant s'abstenir n'affiche aucun nom.

La synchronisation sur un nom N_j est possible si un groupe se forme ; constitué de processeurs ayant affiché exclusivement le nom N_j et si aucun autre processeur n'est en indécision sur ce nom (affichant au moins deux noms dont N_j).

Exemple : 4 processeurs $\{P_0, P_1, P_2, P_3\}$ et 4 noms $\{N_0, N_1, N_2, N_3\}$

	P_0	P_1	P_2	P_3
Initialisation	$N_0\ N_1\ N_2\ N_3$	$N_0\ N_1\ N_2\ N_3$	$N_0\ N_1\ N_2\ N_3$	$N_0\ N_1\ N_2\ N_3$
Synchronisation possible entre P_1 et P_3 sur N_1	N_0 x x N_3	N_1	$N_0\ N_2$	N_1
Synchronisation possible sur N_0 entre P_0 et P_2	N_0		N_0	

II.7.3.1. Réalisation matérielle

Toujours pour des raisons d'efficacité, ces mécanismes ont été confiés au niveau du cablage.

Chaque processeur dispose d'un registre comportant autant de bits qu'il y a de noms de synchronisation. Il émet dans ce registre les noms des groupes auxquels il est candidat.

Une deuxième couche logique permet de détecter les groupes qui se forment et délivrer, selon les critères énoncés précédemment, les signaux d'autorisation de synchronisation éventuelle aux processeurs concernés.

Dans la réalisation matérielle qui nous concerne, nous avons opté pour 4 noms de synchronisation N_0, N_1, N_2, N_3. La table suivante illustre les règles de synchronisation pour ce jeu de 4 noms.

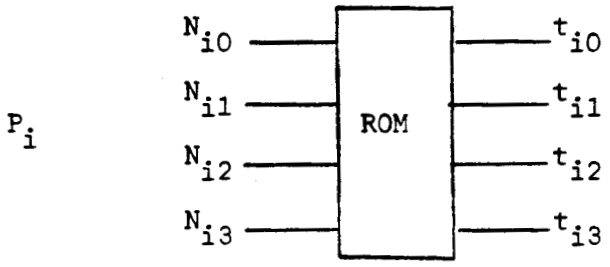
Noms affichés	Autorisations délivées	Commentaires
N_0 N_1 N_2 N_3	t_0 t_1 t_2 t_3	
0 0 0 0	1 1 1 1	Choix final
0 0 0 1	1 1 1 1	Choix final N_3
0 0 1 0	1 1 1 1	Choix final N_2
0 0 1 1	1 1 0 0	Autorise t_0 t_1
0 1 0 0	1 1 1 1	Choix final N_1
0 1 0 1	1 0 1 0	Autorise t_0 t_2
0 1 1 0	1 0 0 1	Autorise t_0 t_3
0 1 1 1	1 0 0 0	Autorise t_0
1 0 0 0	1 1 1 1	Choix final N_0
1 0 0 1	0 1 1 0	Autorise t_1 t_2
1 0 1 0	0 1 0 1	Autorise t_1 t_3
1 0 1 1	0 1 0 0	Autorise t_1
1 1 0 0	0 0 1 1	Autorise t_2 t_3
1 1 0 1	0 0 1 0	Autorise t_2
1 1 1 0	0 0 0 1	Autorise t_3
1 1 1 1	0 0 0 0	N'autorise rien

$\{t_i\}$ = autorisat délivrées

t_i = 1 : autorisat de se synchr niser sur N_i

La réalisation effective pour les 16 processeurs et pour 4 noms s'est faite de la façon suivante :

- La première couche locale aux processeurs comporte des mémoires ROM dans lesquelles sont dupliquées les tables précedentes.



Cette couche aurait pu être réalisée grâce à un P.L.A.

- Les t_{ij} constituent les entrées d'un circuit logique cablé permettant la constitution dynamique des groupes.

- La synchronisation entre deux micro-instructions de 250 ns de cycle est garantie.

La primitive de synchronisation est la suivante :

SET nom de synchro
ATTENDRE : tant que ¬ autorisation aller à ATTENDRE
RESET nom de synchro

II.7.4. Conclusion sur les synchronisations

Les mécanismes de synchronisation tels qu'ils ont été présentées et mis en oeuvre présentent un risque certain d'interblocage résultant notamment du fait qu'un ou plusieurs processus concernés par une action de synchronisation ne passent pas par le rendez-vous ou soient défectueux. De ce fait, ce mécanisme doit être manipulé avec précaution ; il doit donc être utilisé par des concepteurs avertis ou à travers des compilateurs capables de détecter des erreurs de programmation et de matériels.

D'autre part, ces mécanismes conviennent à des contraintes de synchronisation stricte assurant un fonctionnement de type MIMD ou SIMD. On peut définir à partir de là quelques domaines d'applications où de tels mécanismes de synchronisation apportent des avantages :

- Les applications exigeant une répartition parallèle du traitement pour aboutir à des performances élevées. C'est le cas par exemple de processus temps réels exigeant des temps de réponse très brefs incompatibles avec les technologies actuelles.

- Les applications à couplage fort exigeant des coopérations fréquentes.

- Les applications nécessitant la formation dynamique et associative de groupe de processus ; c'est le cas en reconnaissance de formes, intelligence artificiell

II.8. Conclusion générale

Ainsi définie et conçue, l'architecture générale de la machine se compose des blocs fonctionnels suivants :

- Un réseau de 16 processeurs de traitements microprogrammables disposant chacun de sa mémoire de microprogrammes (8 K mots de 16 bits).

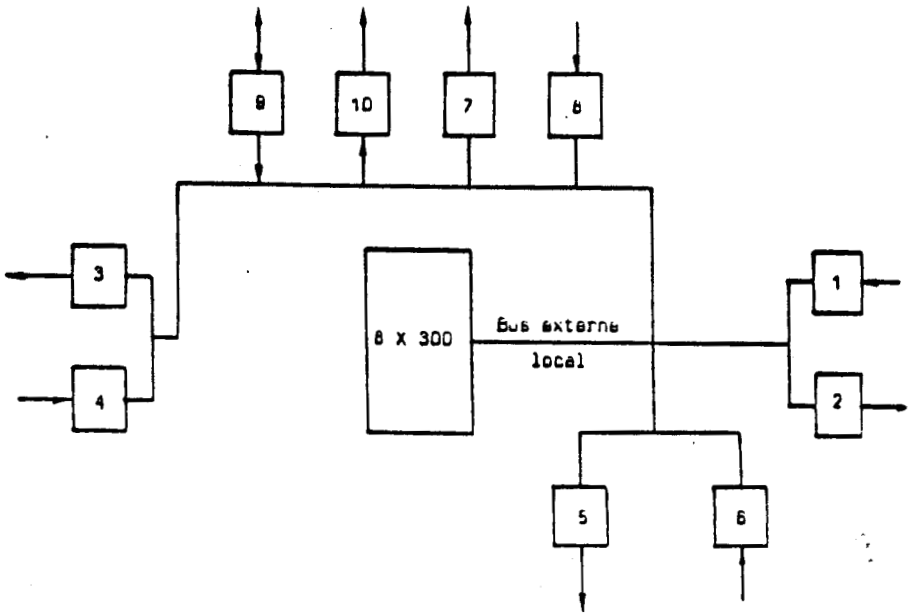
- Un réseau de communication et de routage véhiculant les informations entre les différents sites.

- Un réseau de synchronisation assurant les cohérences temporelles et d'exécution entre les constituants du système.

- Le système est contrôlé et supervisé par un processeur maître assurant en plus la fonction d'entrée/sortie.

- Une horloge commune pilote tout le système assurant ainsi une évolution synchrone de l'ensemble des micro-instructions.

Dans la suite de cette étude nous allons détailler une application de cette architecture dans le cas de la synthèse d'images.



Registres :

1, 2, 3, 4 : communication avec les processeurs voisins ;

5, 6 : communication avec la mémoire d'image ;

7 : scrutation ;

8 : diffusion ;

9, 10 : synchronisation.

Fig. II.11. : Configuration d'un processeur.

SOMMAIRE DU CHAPITRE III

III. LE SYSTEME GLOBAL

III.1. Introduction

Les besoins d'applications spécialisées impliquent l'existence d'outils matériels et logiciels spéciaux assurant les traitements ainsi que l'interfaçage avec des opérateurs humains. Si les traitements internes sont confiés à des unités de traitements, de calculs et de stockage sous une forme condensée et codée compatible avec les organes logiques, il n'en va pas de même des unités d'interface et de communication qui doivent assurer la transduction de manière à présenter à l'opérateur humain des informations "compréhensibles" sous d'autres formes codées.

Ce chapitre décrit les éléments matériels et logiciels qui constituent l'environnement du processeur graphique. Ces éléments permettent l'exploitation du système multiprocesseur de façon aisée ainsi que la possibilité de développer des microprogrammes de traitement graphique. La configuration globale du système est la suivante :

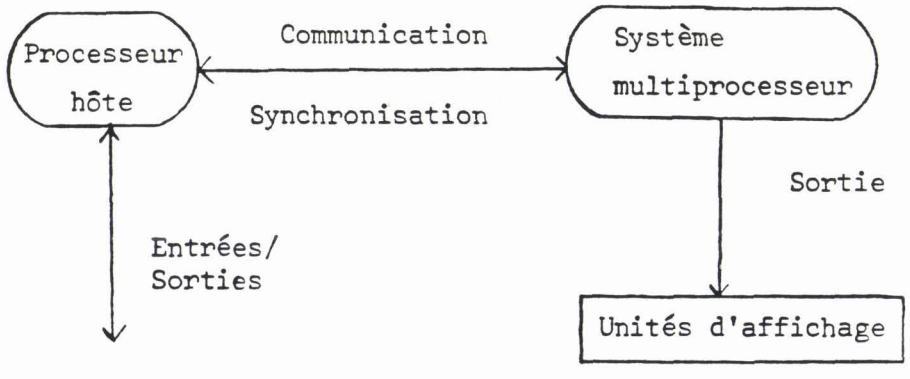


Fig. III.1. : Configuration globale

III.2. Le processeur hôte

Le rôle du processeur hôte est donc de décharger le système multiprocesseur des tâches logicielles ayant pour résultat le contrôle ainsi que les demandes d'exécution de séquences de traitement implantées dans les différentes mémoires de microprogrammes. Ce processeur assure aussi le dialogue avec l'utilisateur.

Les critères de choix de ce système sont les suivants :

- Grande modularité permettant une extension du système.

- Existence d'une gamme étendue d'outils de développement.

- Simplicité de mise en oeuvre et d'utilisation.

- Coût modéré.

Ces considérations et bien d'autres nous ont amenés à choisir les systèmes multibus d'intel (INT, ZEN).

III.2.1. Description du concept multibus

Ce concept permet la réalisation de systèmes multiprocesseurs se partageant un bus commun ; le multibus. Chaque carte processeur dispose néanmoins des ressources et bus locaux lui permettant un fonctionnement autonome. Des dispositifs de gestion de priorités permettent l'attribution du multibus à une carte évitant ainsi les conflits d'accès à ce dernier.

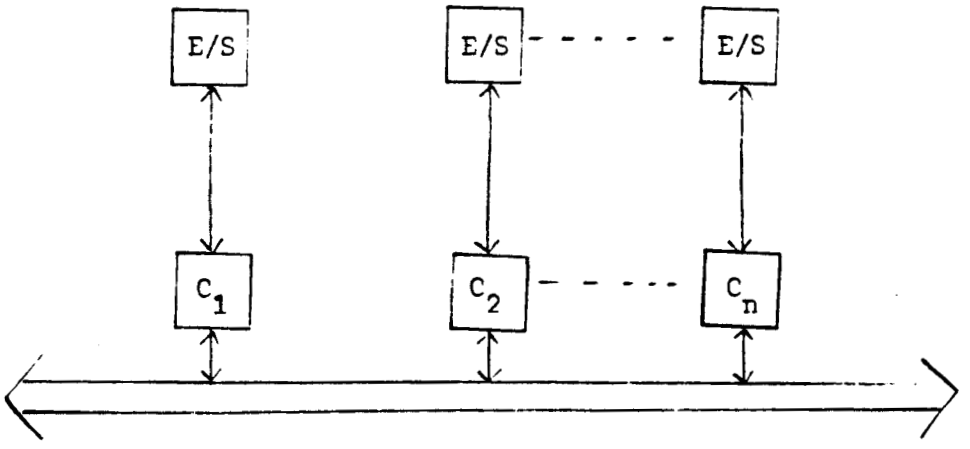


Fig. III.2. : Systèmes multibus.

Des interfaces d'entrée-sortie assurent la connexion avec des périphériques.

Les avantages d'un tel système sont nombreux :

- Possibilité d'extension du système par la connexion directe sur le multibus d'une ou de plusieurs cartes.

- Possibilité de remplacer directement une carte défectueuse par une autre.

- Existence d'un grand nombre de cartes pouvant servir de cartes CPU, de cartes mémoires, ou de cartes contrôleur d'entrée-sortie.

- Existence de systèmes de développement complets tournant sur ces cartes.

- Systèmes à base de 8080, 8085, 8086 ... (8 ou 16 bits) donc relativement p formants.

III.2.2. La configuration adoptée

Le système hôte se compose d'une carte maître basée autour du microprocesseur 8085 qui assure la gestion du multibus ainsi que les interfaces suivantes :

- Communication et synchronisation avec le processeur maître du système multiprocesseur.

- Gestion d'une console alpha-numérique.

- Contrôle et gestion d'une imprimante.

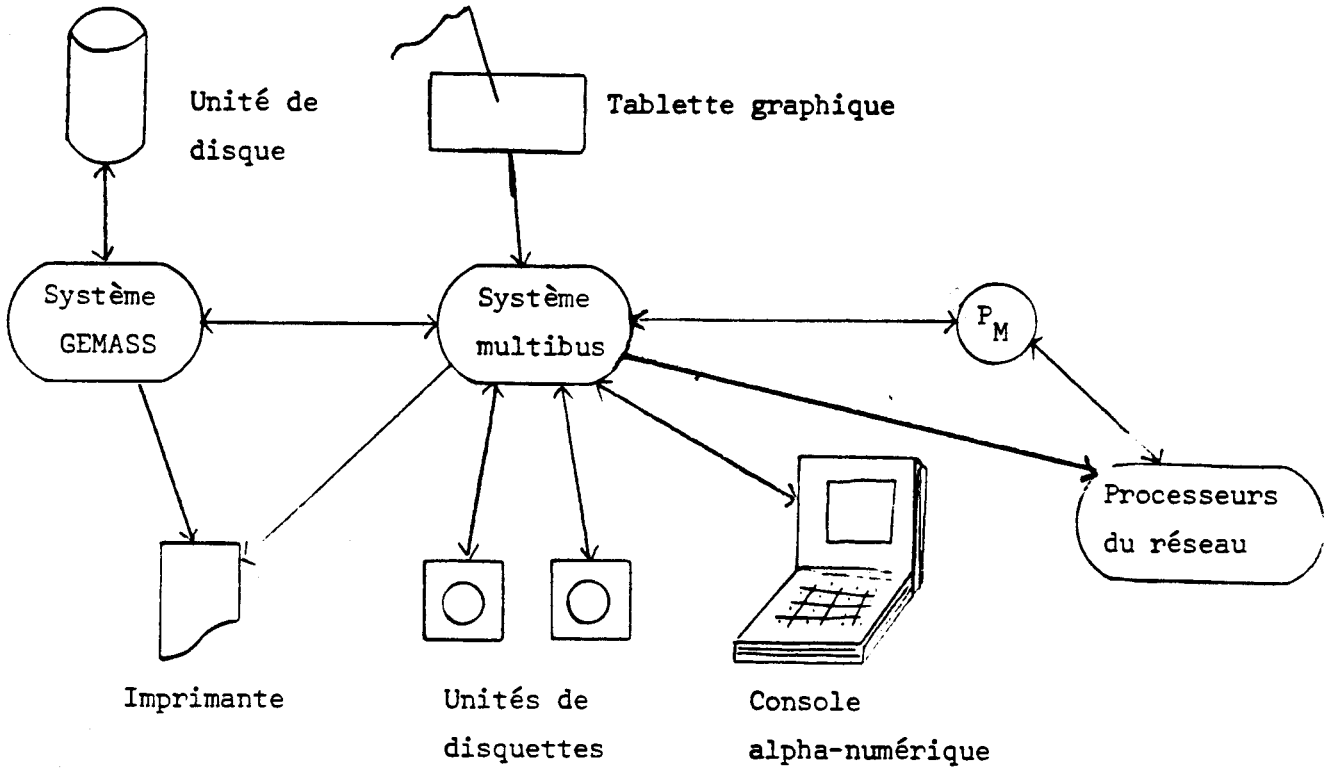
- Communication et synchronisation avec un système de gestion de mémoire de masse (le système GEMASS (CAR 82)).

- Contrôle d'une tablette graphique.

Deux cartes esclaves sont connectées sur le multibus :

- Une carte contrôlant deux unités de disquettes (8 pouces double densité).

- Une carte assurant le chargement de microprogrammes dans le système multiprocesseur.



Les communications entre le processeur hôte et le système multiprocesseur se font à deux niveaux :

- Chargement de séquences de microprogrammes. En effet, dans le but de développer les logiciels et pour ne pas avoir des séquences de traitements figées, les différentes mémoires ont été implémentées à l'aide de R.A.M. (cf. III.4.).

- Exploitation du système multiprocesseur ; dans ce cas une liaison parallèle 8 bits permet d'interfacer le processeur hôte avec le processeur maître. Le fonctionnement se fait selon un principe de producteur-consommateur où le 8085 est producteur ; P_M le consommateur. Une bascule (B) assure la synchronisation entre les deux partenaires :

P8085

```
Prod : Production d'un octet ;
       B := Plein ;
Att : Si B = vide alors aller à Prod ;
      sinon aller à att fsi ;
```

P_M

Att : Si B = PLEIN alors aller à CONS ;
sinon aller à att fsi ;

CONS : CONSOMMER un octet ;
B := VIDE ;
aller à att ;

III.2.3. Outils logiciels

Le processeur hôte offre de nombreux outils logiciels indispensables au développement et à la mise au point des programmes de l'utilisateur. A cet effet, ce processeur fonctionne sous le système opératoire ISIS (INTEL) et permet dans la configuration actuelle :

- La programmation en assembleur et en langage machine 8085.

- La programmation en PLM 80.

- La gestion de fichiers (gestion de deux unités de disquettes, liaison parallèle avec une mémoire de masse).

- La production de micro-codes 8 × 300 (grâce à un CROSS-ASSEMBLEUR 8 × 300).

- La programmation en langage FORTH.

- Des fonctions diverses de mises au point et de développement de programmes telles que éditions de textes, diagnostics d'erreurs,...

- Etc...

III.3. Le processeur trace (GOM 81)

Ce module est rendu indispensable pour la mise au point des différents micro-programmes qui se déroulent sur les sites. Son rôle est de visualiser à

la demande l'activité des 17 processeurs à l'échelle de la micro-instruction selon divers modes de trace :

- Tracer tous les processeurs en même temps.

- Tracer deux processeurs en même temps.

- Démarrer la trace sur une micro-instruction précise.

- Démarrer la trace sur une adresse précise.

- Tracer certains types particuliers d'opérations.

Ce processeur est conçu à partir des mêmes éléments que les processeurs du réseau (à base de famille 8 × 300).

La simplicité de mise en oeuvre ainsi que l'homogénéité des logiciels ont motivé ce choix.

L'affichage se fait sur un écran alpha-numérique adressable.

Le schéma synoptique est celui de la figure III.3.

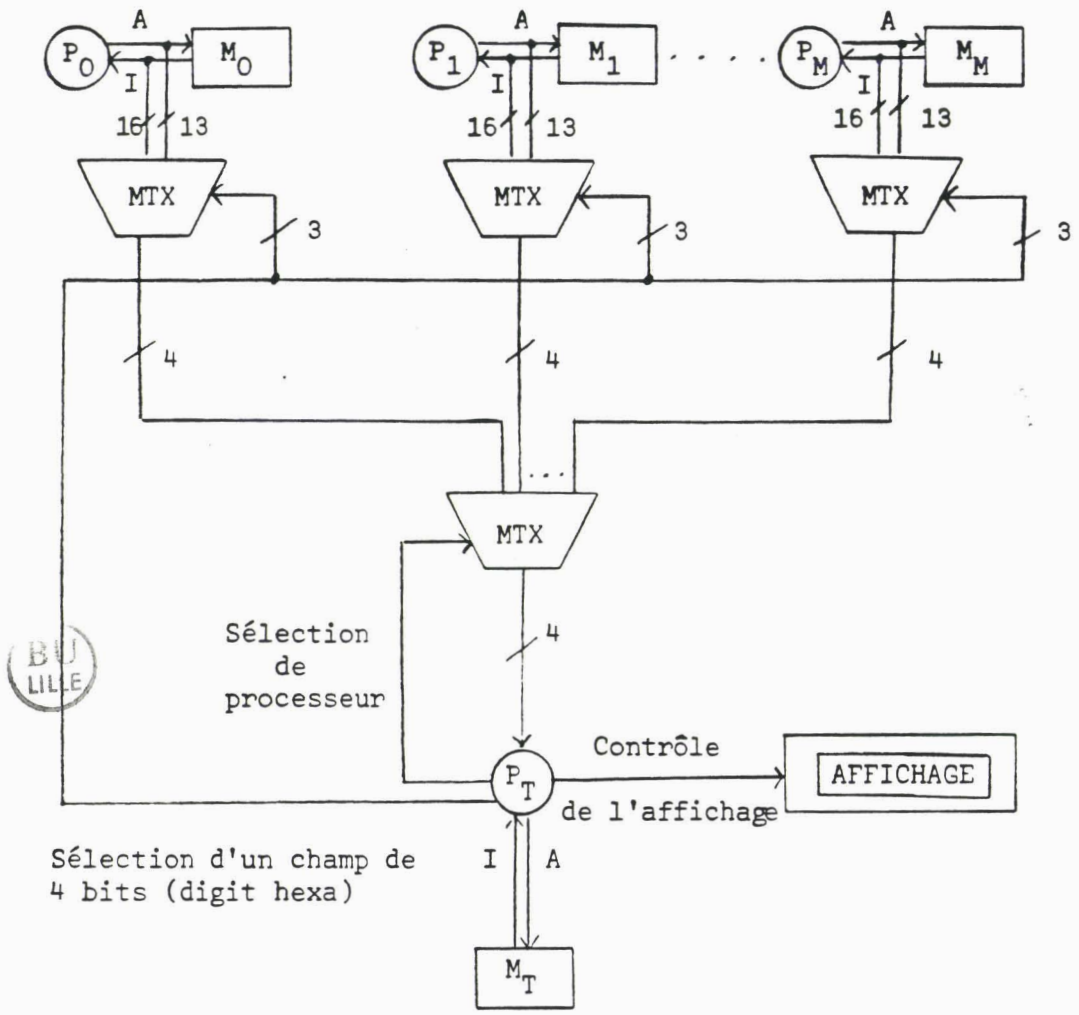


Fig. III.3. : Processeur trace.

P_i : Processeur i. P_T : Processeur trace

M_i : Mémoire de microprogramme associée à P_i.

MTX : Multiplexeur

A : Adresse (sur 13 bits).

I : micro-instruction.

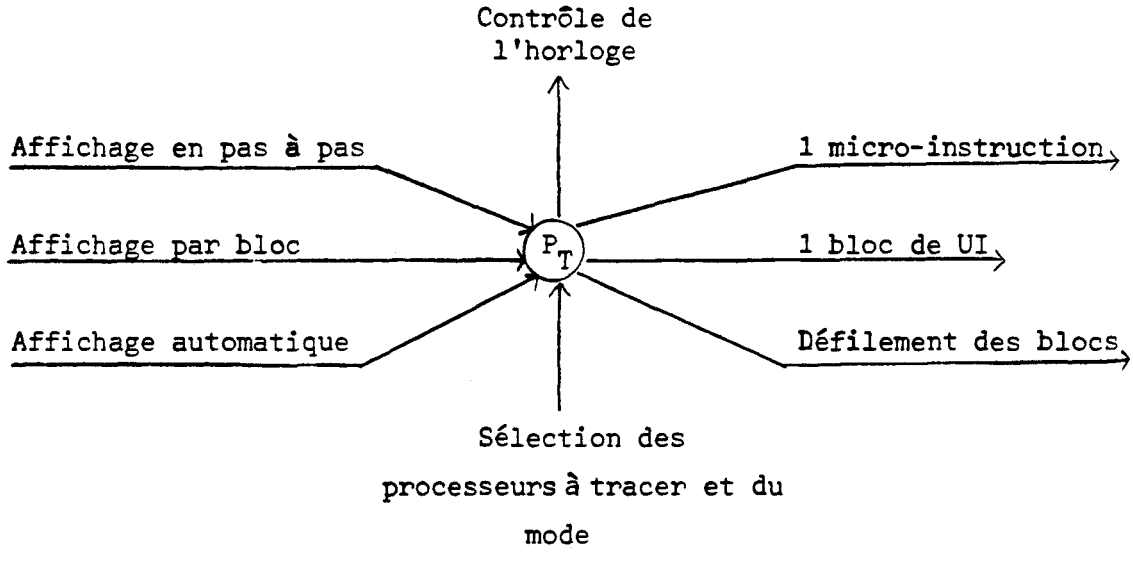
En mode trace, P_T contrôle l'horloge des processeurs par l'émission d'une horloge programmée permettant ainsi d'observer l'évolusion des différentes micro-instructions (Affichage en assembleur 8 × 300).

L'affichage peut se faire de trois manières différentes :

- Affichage en pas à pas : Les micro-instructions sont visualisées une par une à la demande. (Bouton poussoir au pupitre).

- Affichage par bloc : Les micro-instructions sont affichées par bloc (un bloc correspond à la taille de l'écran 10 lignes), à la demande.

- Défilement : L'affichage se fait par blocs de façon automatique.



III.4. L'automate de chargement

Le système multiprocesseur pouvant être dédié à des fins diverses suivant le logiciel d'une part, et le fait que les différents microprogrammes destinés à tourner sur la machine nécessitent, dans un premier temps, des tests de mise au point et d'optimisation nous a conduit à concevoir les mémoires de microprogrammes à l'aide de RAM au lieu de ROM ou de REPROM dans lesquelles les microprogrammes auraient été figés et toutes modifications de ceux-ci difficiles et très coûteuses en temps.

Ceci nous conduit aussi à réaliser un automate de chargement de ces R.A.M. Ce chargement pouvant se faire de façon manuelle (à partir du pupître) ou à l'aide d'un programme tournant sur le 8085. Cette solution est de loin plus intérressante dans le cas du chargement de plusieurs micro-instructions.

L'automate de chargement doit gérer les signaux suivants :

- Sélection de la mémoire à charger (une parmi 17, le processeur trace exécute des micro-programmes figés en ROM).

- Sélection de l'adresse de chargement.

- Envoi du micro-code à charger (mots de 16 bits).

- Envoi de la commande d'écriture.

III.4.1. Réalisation matérielle

La réalisation matérielle doit prendre en compte l'optimisation de la longueur du chemin de données. Pour ce faire l'automate a été conçu de la façon suivante :

- L'adresse de la micro-instruction à charger est délivrée par le microprocesseu concerné qui agit dans ce cas comme un générateur d'adresses.

L'automate délivre un signal de remise à 0 ($\overline{\text{RESET}}$ du 8 × 300) ; le passage d'une adresse à la suivante se faisant alors par action sur l'horloge. Ceci est rendu possible étant donné que le 8 × 300 n'exécute des instructions de rupture de séquence que si son bit de poids fort est à 1 ; il suffit donc de mettre à 0 ce bit pendant le chargement.

- Un code à charger est envoyé octet par octet, l'automate élabore alors deux signaux d'écriture mémoire ; l'un servant à écrire dans le poids faible (W_f), l'autre dans le poids fort (W_F).

III.5. Le convertisseur

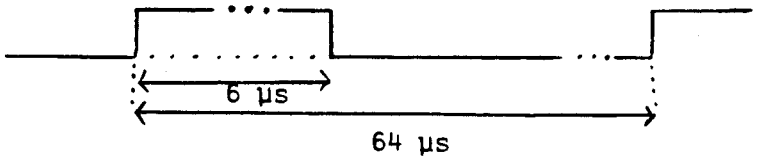
Le rôle du convertisseur est d'assurer l'affichage des images sur le visuel. La visualisation se fait sur un écran de télévision couleur standard. Nous rappelons les caractéristiques principales de ce matériel :

- Le balayage : Le balayage se fait de la gauche vers la droite et de haut en bas en 40 ms (temps de balayage d'une trame en mode non entrelacé). En fait, pour diminuer le scintillement ; l'écran est balayé en deux demi-trames entrelacées d'une durée de 20 ms chacune.

- Les signaux de gestion de l'écran :

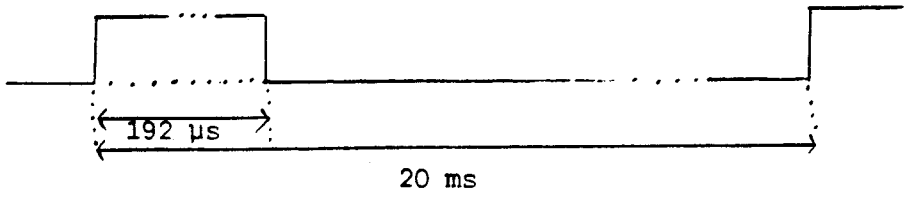
Ces signaux servent au maintien du balayage et à l'affichage de la couleur sur l'écran.

(a) - Le signal de synchronisation ligne (S_ℓ) :



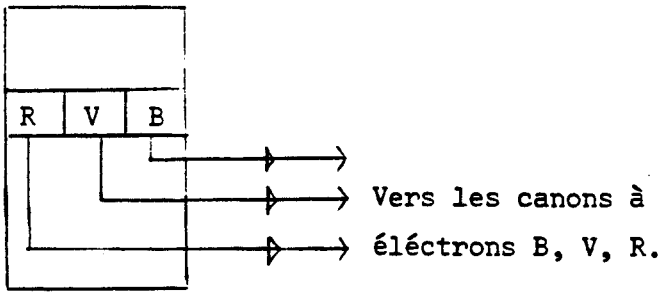
Une ligne dure 64 µs.

(b) - Le signal de synchronisation Trame (S_T) :



(c) - <u>Le signal vidéo</u>

Ce signal correspond à l'émission "du code de la couleur" à visualiser sur l'écran. La conversion en grandeurs analogiques agissant sur les trois canons à électrons du téléviseur est réalisée grâce à des convertisseurs digitaux-analogiques (C.D.A.).



CDA

Un pixel est émis toutes les 125 ns.

(d) - Cadrage de l'image dans l'écran

Théoriquement l'écran peut afficher 625 lignes de 512 points chacune (en pratique le nombre de points affichables est toujours inférieur à ce nombre ceci étant dû au masque). L'image mémoire ne faisant que 256 lignes de 256 points chacune doit être cadrée par rapport à ce dernier.

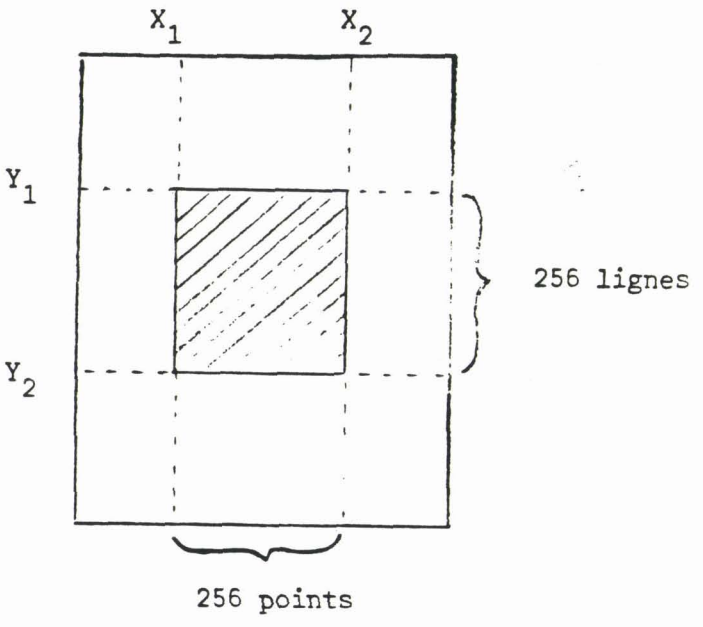


Fig. III.4. : Cadrage de l'image dans l'écran.
L'image est visualisée dans la surface hachurée.

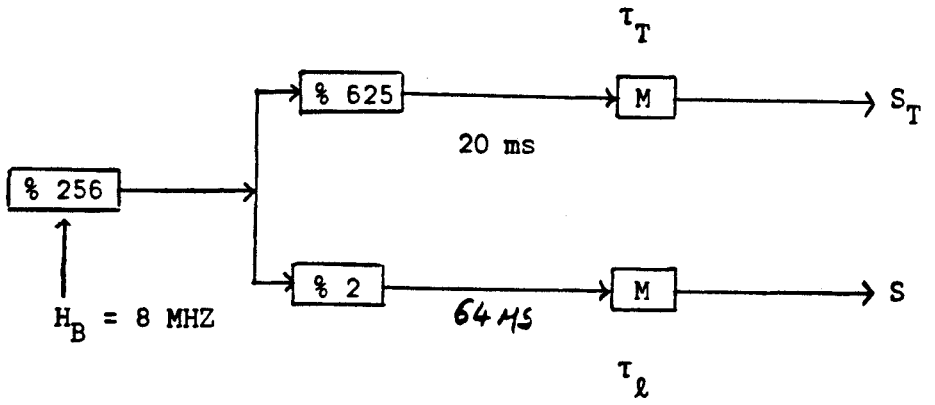
La durée de l'affichage d'une image est donc :

256 × 256 × 125 = 32 ms × 256 = 8,192 ms.

Soit un temps mort pour une demi-trame de 20 ms - 8,192 ms ≃ 11,8 ms soit plus de 50 %.

(e) - Elaboration de ces signaux

Le convertisseur élabore ces signaux grâce à une horloge de base de 125ms de période (8 MHZ) selon le schéma suivant :



M : Monostable

τ_i : Constantes de temps pour le maintien des impulsions S_ℓ et S_T ($i \equiv \ell$ ou $i \equiv T$).

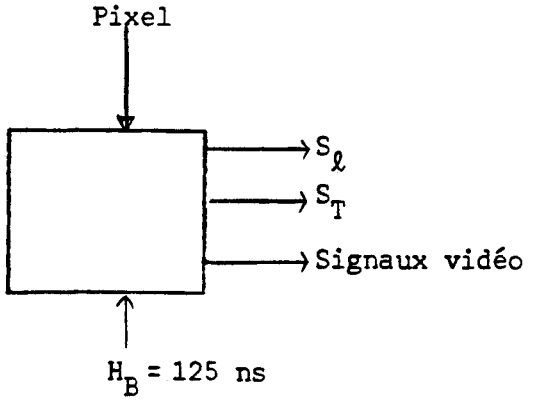


Fig. III.5. : Interface convertisseur - Ecran.

III.5.1. La logique d'accès direct en mémoire d'images A.D.M.

L'A.D.M. est une logique permettant l'entretien de l'image affichée par accès direct en mémoire d'images. Ceci suppose que cet accès doit être le plus rapide possible afin de ne pas pénaliser les performances du système multi-processeur.

Ceci nous a conduit à définir deux registres R_A et R_B de 16 × 8 bits chacun, fonctionnant en bascule et assurant le rafraîchissement de l'écran. L'accès en mémoire se fait donc par blocs de 128 bits. Une commande d'arrêt commune aux processeurs (signal commandant directement la broche $\overline{HALT}$ du 8 × 300) permet l'arrêt de ceux-ci, les registres de lecture mémoire sont inhibés durant ce cycle d'accès. Pendant qu'un registre se charge en parallèle le contenu de l'autre est visualisé par une série de décalages de 8 bits vers l'écran toutes les 125 ns et vice-versa.

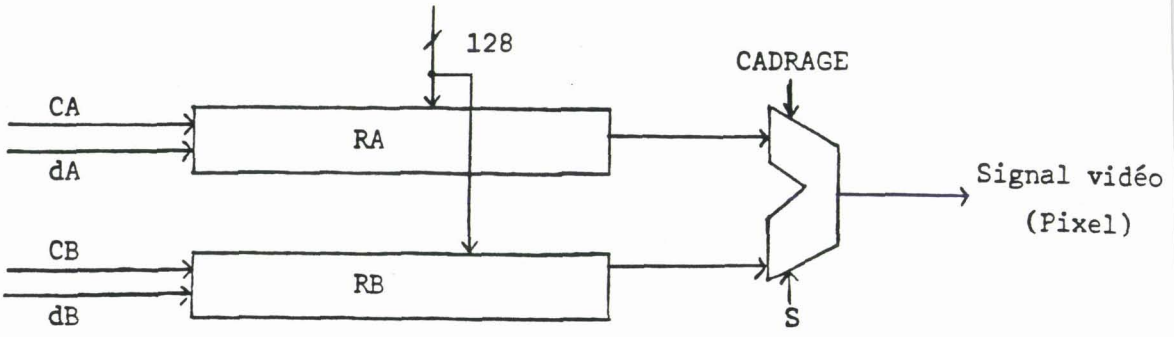
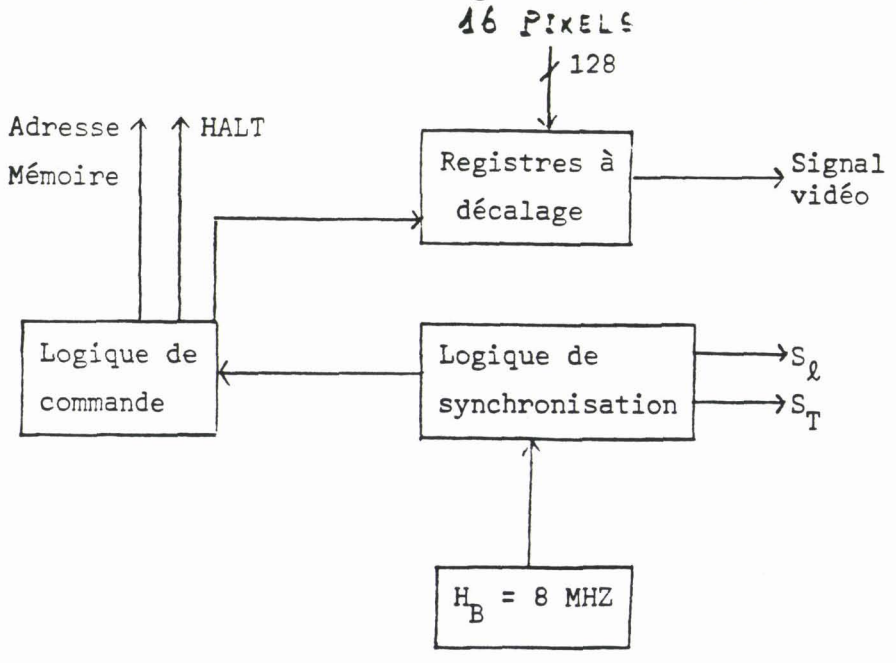


Fig. III.6. : Les registres à décalage.

CA, CB : Commandes de chargement parallèle de RA, RB.
dA, dB : Commandes de décalage vers l'écran de RA, RB.
S : Sélection de la voie à afficher.
CADRAGE : Signal de cadrage de l'image.

Le schéma fonctionnel général de l'A.D.M. est le suivant :



III.5.2. La table de couleurs

Un autre élément très important du convertisseur est la table de couleurs. Cette table s'intercale entre la mémoire de trame et l'écran. La valeur des pixels n'est plus, alors, utilisée directement pour attaquer les C.D.A. mais sert d'index dans une table dite table des couleurs, à l'emplacement désigné se trouve la valeur qui servira de signal vidéo.

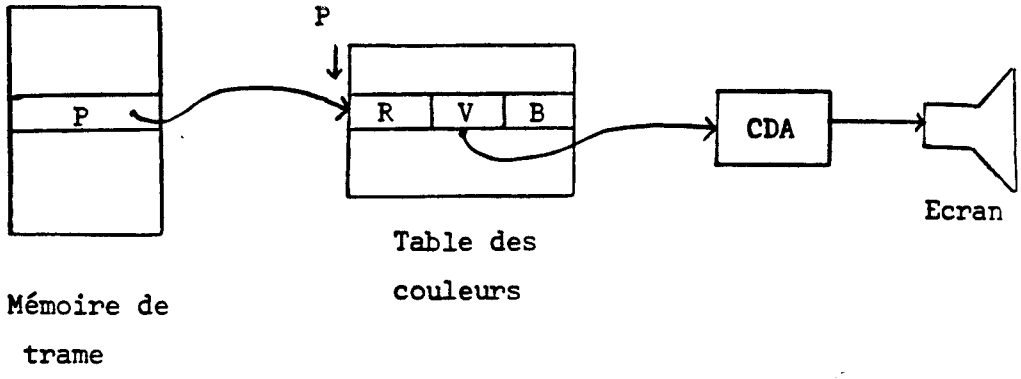


Fig. III.7. : Table des couleurs.

L'un des principaux intérêts de ces tables est qu'elles permettent d'avoir une gamme étendue de couleurs tout en limitant la taille de la mémoire d'images.

Si n est le nombre de bits par pixel dans la mémoire d'images on arrive a avoir des pixels codés sur Q bits ; avec Q > n ; en intercalant une table de 2^n mots de Q bits.

Ces tables permettent, par la modification de leur contenu, de changer l'aspect (couleur) de l'image stockée dans la mémoire de trame. Cette deuxième remarque montre l'intérêt que peuvent avoir ces tables dans la synthèse et l'animation d'images.

Supposons, à titre d'exemple purement académique, que l'on veuille visualiser un carré qui effectue le parcours matérialisé par les flèches sur la figure III.8.

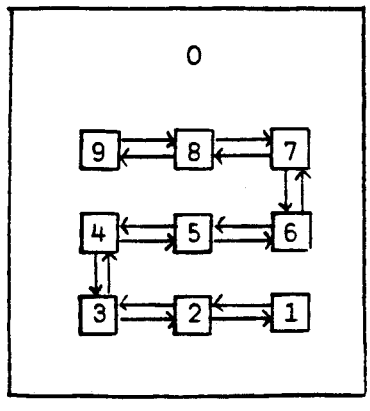


Fig. III.8. : Exemple d'animation. Les numéros à l'intérieur des carrés correspondent aux codages des pixels associés.

La figure III.9. montre comment doit être chargée la table des couleurs associée à chaque étape d'animation. Nous supposons que le carré est rouge et le fond vert.

Adresse T.C.	Contenu en fonction de l'étape d'animation												
	1	2	3	4	5	6	7	8	9	10	11	12	.....
0	V	V	V	V	V	V	V	V	V	V	V	V	
1	R	V	V	V	V	V	V	V	V	V	V	V	
2	V	R	V	V	V	V	V	V	V	V	V	V	
3	V	V	R	V	V	V	V	V	V	V	V	V	
4	V	V	V	R	V	V	V	V	V	V	V	V	
5	V	V	V	V	R	V	V	V	V	V	V	V	
6	V	V	V	V	V	R	V	V	V	V	V	R	
7	V	V	V	V	V	V	R	V	V	V	R	V	
8	V	V	V	V	V	V	V	R	V	R	V	V	
9	V	V	V	V	V	V	V	V	R	V	V	V	
⋮													

Fig. III.9. : Table de couleurs correspondantes.

La table des couleurs subit une modification avant chaque étape d'animation, ceci est génant car elle sert à l'entretien de l'écran. On trouvera dans (FER 83) une solution à ce problème par l'emploi de deux tables de couleurs commutables; pendant que l'une sert au rafraîchissement de l'écran, l'autre peut être modifiée et vice-versa (Fig. III.10). Certaines précautions doivent être prises ; la commutation de tables n'étant possible qu'après l'affichage d'un demi-trame (synchrone de S_T) par exemple.

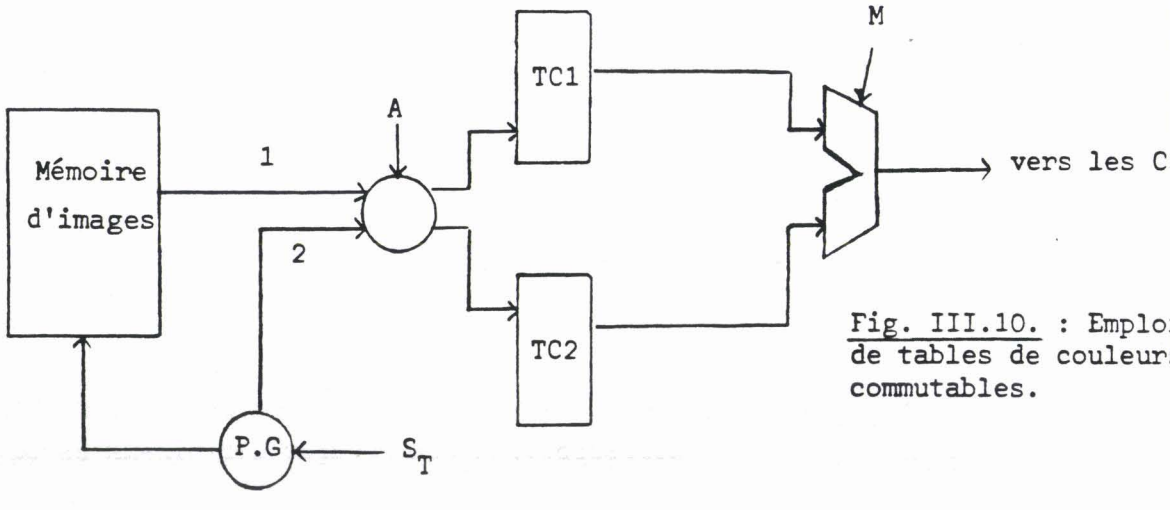


Fig. III.10. : Emploi de tables de couleurs commutables.

A : Aiguillage : 1 → TC1 ou 1 → TC2
2 → TC2 ou 2 → TC1.

M : Multiplexage : TC1 → CDA ou TC2 → CDA.

P.G : Processeur graphique.

S_T : Signal de synchro trame.

TC1, TC2 : Tables de couleurs.

Il est aussi possible d'utiliser ces tables à d'autres fins ; nous en citerons quelques-unes :

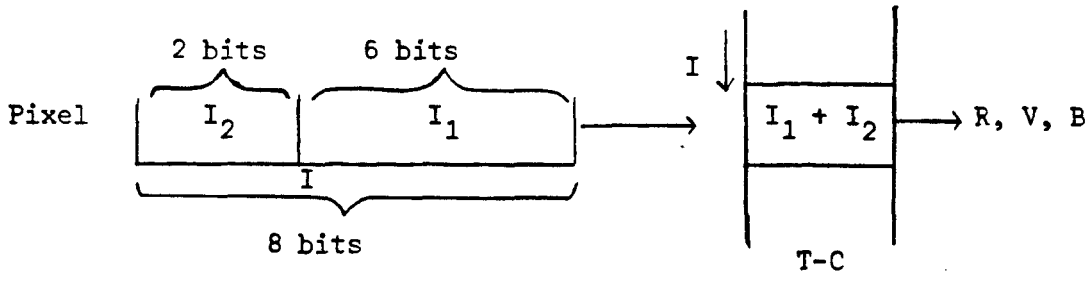
- Combinaison d'images

Les codes de la mémoire d'images sont considérés comme appartenant à des images différentes ; la combinaison consiste en une série d'opérations sur ces images. Le résultat de ces opérations servira d'index dans la table des couleurs.

Exemple : Supposons que les pixels en mémoire d'images soient codés sur 8 bits
Ces pixels peuvent être considérés de façons différentes :

- deux images codées l'une sur deux bits et l'autre sur six bits.
- huit images codées chacune sur un bit.
- etc...

L'exemple de combinaison le plus simple est sans doute l'addition d'images. Nous aurons dans le premier cas



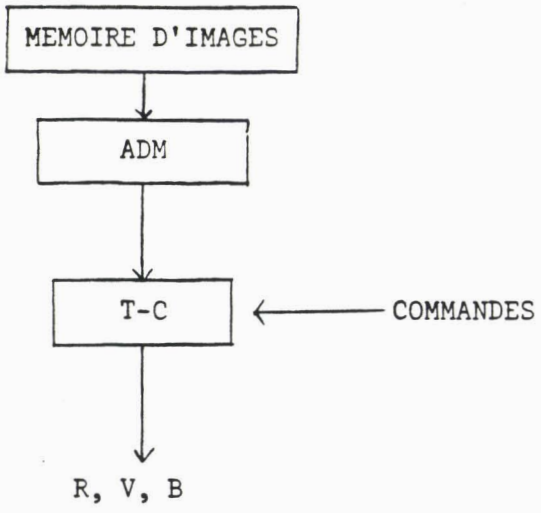
- Traitements de surfaces cachées :

Comme dans le cas précédent, un pixel peut être considéré comme appartenant à plusieurs images différentes. Pour le traitement de surfaces cachées ; à chaque partition du pixel est attachée une priorité déterminant un plan d'affichage par rapport à un observateur ; un point ne peut s'afficher que si aucun autre plus prioritaire ne le "cache". Ceci donne ainsi la possibilité de traiter des images à trois dimensions.

- Autres applications

Il existe beaucoup d'autres applications des tables de couleurs nous ne les détaillons pas ici. Signalons les applications matérielles telles que les transformations TIS/TBN → RVB, etc...
Il faut remarquer aussi que l'on trouve parfois dans la littérature une distinction entre les tables des couleurs qui délivrent des grandeurs R, V, B et les tables de compensation qui, elles, délivrent un index dans les tables des couleurs.

Le schéma fonctionnel du convertisseur est le suivant :



III.6. Conclusion

Nous avons considéré dans ce chapitre les moyens qui constituent l'environnement du système multiprocesseur. Il faut souligner que l'efficacité et la performance de ces moyens ont une grande implication sur celles du système. Le convertisseur va agir directement sur les processeurs de traitement en les arrêtant pour accéder en mémoire d'images via la logique A.D.M, cet arrêt doit être le plus court possible et dépend en fin de compte de la logique A.D.M. et du temps d'accès en mémoire d'images (actuellement on trouve des RAM 2K octets pouvant avoir jusqu'à 50 ns de temps d'accès pour un coût acceptable). D'autre part les temps d'accès aux tables des couleurs doivent être rapides, afin de permettre un entretien correct de l'écran.

SOMMAIRE DU CHAPITRE IV

IV. LE NOYAU DE PRODUCTION DE MICROPROGRAMMES PARALLELES

IV.1. Introduction

La production de microprogrammes parallèles destinés à être exécutés sur les différents processeurs du système suppose que soient prise en compte des contraintes de coopération, de communication et de synchronisation.

Plusieurs processus peuvent coopérer à l'exécution d'une tâche et doivent donc s'échanger des informations concernant les données sur lesquelles ils effectuent des traitements ainsi que l'état respectif de leur évolution.

Nous écartons à priori la solution qui consiste à laisser à l'usager le soin de rédiger lui-même ses séquences de micro-programmes parallèles car elle présente les inconvénients suivants :

- Le programmeur devra prévoir tous les cas possibles pouvant intervenir lors de l'exécution de ses micro-programmes et il doit donc les rédiger en conséquenc

- L'architecture de la machine est d'un ordre de complexité élevé, le programmeu doit être doté d'une formation spéciale lui permettant de maîtriser cette architecture et d'arriver ainsi à optimiser ses micro-programmes.

- Plusieurs fonctions du système ne peuvent être maniées sans risques certains de détériorer le fonctionnement du système (risque d'interblocage dans le cas d'une mauvaise programmation des outils de synchronisation), on ne peut accéder à ces fonctions qu'à travers des séquences spécifiques assurant la cohérence et le contrôle.

- Enfin, la productivité de microprogrammes est extrêmement faible : un programmeur expérimenté, bien au fait de l'architecture de la machine ne produit que 2 à 10 micro-instructions par heure.

La solution adoptée consiste en l'écriture d'un certain nombre de macro-commandes spécifiques à l'application (dans notre cas ce sont des macro-commandes graphiques orientées vers l'animation d'images).

Une macro-commande étant une séquence de micro-programmes parallèles réclamant un certain concours à tout ou partie des processeurs du réseau. Ces macro-commandes sont produites grâce à un interpréteur écrit dans un langage de haut niveau interprétable et rapide ; nous avons choisi le langage FORTH (voir Chapitre V).

Pour ce faire le contrôle est hiérarchisé à plusieurs niveaux :

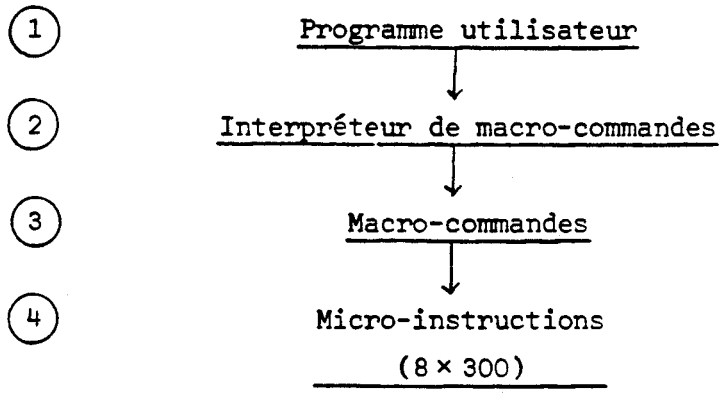


Fig. IV.1. : Les niveaux de contrôle.

- Le niveau (1) correspond au programme utilisateur, ce programme est pris en charge par l'interpréteur de macro-commande au niveau du processeur hôte (P8085).

- Le niveau (2) produit deux sortes d'informations :

  * Des codes identifiant les macro-commandes. Ces codes sont transmis par le processeur hôte au processeur maître P_M selon un principe de producteur-consommateur.

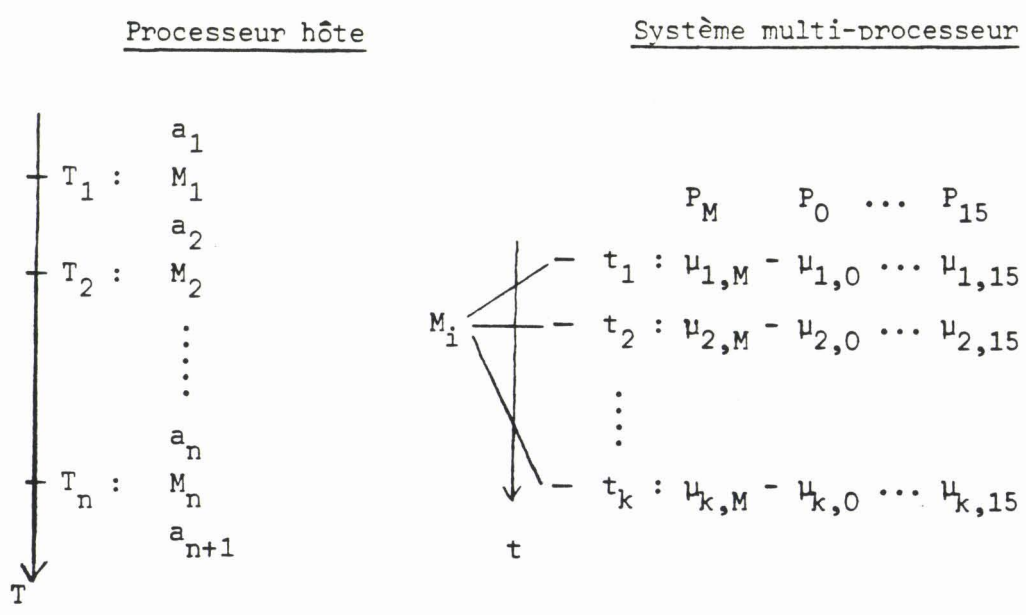
  * Des fonctions de contrôle qui sont exécutées localement (boucles, alternatives,...).

- Le niveau (3) correspond aux macro-commandes ; après réception des codes spécifiant les macro-commandes à exécuter, le processeur maître émet les consignes nécessaires à l'exécution de celles-ci vers les processeurs du réseau.

- Le niveau (4) correspond à l'exécution des séquences parallèles de micro-instructions exécutées par les différents processeurs du système multiprocesseu (P_M, $P_0 \ldots P_{15}$).

La figure suivante montre plus finement ce principe où $a_j = f(M_1, M_2, \ldots$ structure de contrôle permettant la structuration des programmes utilisateurs.

M_i : Macro-commande dont le corps correspond à l'exécution parallèle de K micro-instructions sur les différents sites du système multiproce seur.



Le séquencement temporel se définit donc à deux niveaux :

- Le niveau T_i haut niveau de cadencement défini par l'interpréteur de macro-commandes. Nous remarquons que celles-ci n'ayant pas obligatoirement les mêmes durées, ces temps sont différents.

- Le niveau t_j correspond à des cadencements de bas niveau, le temps est défini dans ce cas par l'horloge générale du système multiprocesseur, les t_i ont une même durée (voir Chapitre II) grâce aux caractéristiques techniques de la machine.

$\mu_{\ell,k}$: micro-code 8 × 300 exécuté au temps t_ℓ par le processeur k(k = 0, 1, 2,..., 15, M).

A ce stade, nous pouvons distinguer deux niveaux de programmation :

* Une programmation système produisant des micro-programmes parallèles à l'aide de micro-instructions de base.

* Une programmation d'application produisant des programmes à l'aide des macro-commandes.

IV.2. Les macro-commandes

Une macro-commande est une directive globale réclamant un certain concours à l'ensemble des processeurs. D'une manière générale une macro-commande M_i est identifiée de la façon suivante (équivalent au point de vue syntaxe à un assembleur) :

$$\text{codop}, \{p_i\}$$

où codop est un code spécifiant la macro-commande et $\{p_i\}$ est un ensemble de paramètres pouvant être de différentes natures :

- Numéros des processeurs concernés par la macro-commande.

- Adresses dans la mémoire de données.

- Valeurs spécifiant par exemple des couleurs, des incréments d'adresses ou des informations quelconques.

- Valeurs désignant des objets.

Cette notion est très importante car c'est en fait le jeu de macro-commandes qui va spécialiser la machine.

Vue à ce niveau, la machine devient un processeur classique :

- Exécutant des processus séquentiels.

- Doté d'un jeu d'instructions spécifique (les macro-commandes) qui peut varier grâce à la microprogrammation.

Le parallélisme n'apparaît donc absolument plus à ce niveau et il est donc transparent à l'utilisateur.

Nous décrivons dans ce qui suit un jeu de macro-commandes intervenant en synthèse d'images, cette présentation n'est pas exhaustive mais permet de montrer comment s'effectue l'exploitation du multi-processeur.

La détermination d'un répertoire de macro-commandes graphiques doit répondre à certaines conditions et contraintes de leur utilisation :

- Sur quoi vont-elles agir ? Cela revient à définir les objets sur lesquels elles vont agir.

- Quels seront leurs rôles ? i.e. prévoir toutes les macro-commandes nécessaires pour effectuer les commandes de l'utilisateur.

Une réponse générale à ce deuxième point est difficile à donner car chaque application privilégie certaines fonctions spéciales (ex. : précision, netteté... en C.A.O, grande gamme de couleurs... en peinture, etc...).

Ces contraintes sont à rapprocher de celles imposées aux constructeurs d'unités centrales devant offrir un répertoire d'instructions répondant à un cahier de charges donné.

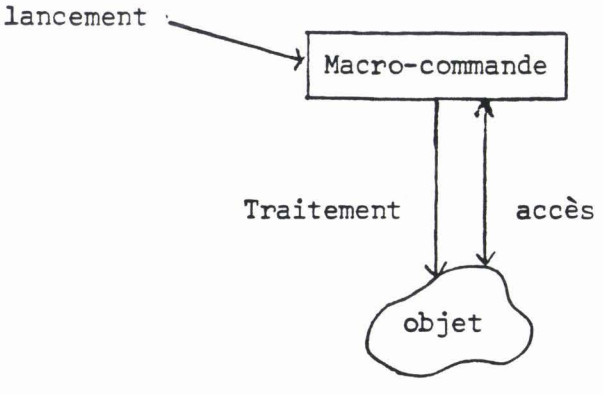


Fig. IV.2. : Macro-commande.

IV.2.1. Les objets manipulés

Les macro-commandes déterminent le noyau de synthèse et d'animation d'images. Celles-ci doivent être très générales pour permettre des applications diverses. Nous distinguerons par la suite les macro-commandes qui, elles, interviennent au niveau de la mémoire d'images des commandes intervenant au niveau du convertisseur et exécutées par le processeur hôte.

D'autres modules microprogrammés existent et permettent l'exécution de fonctions de base ; nous aborderons ces modules un peu plus loin dans ce chapitre ; signalons tout de même qu'il s'agit de modules permettant la synchronisation, l'exploitation de la mémoire d'images...

Nous considérons divers objets dans le système, ces objets peuvent être des sous-ensembles de la mémoire d'images, des listes de valeurs ou des données concernant le convertisseur.

a - Les objets concernant la mémoire d'images

L'accès en mémoire d'images peut se faire de différentes façons selon diverses considérations d'adressages. Dans le but de compresser le code des macro-commandes nous avons défini les types d'objets suivants, ces choix ne sont, bien entendu, pas exhaustifs :

a-1 - La tranche (GRA 80)

On appelera tranche (z_i, z_j) l'ensemble de toutes les zones de la mémoire d'images de z_i à z_j.

tranche (z_i, z_j) ::= $<\{z_i, z_{i+1}, \ldots, z_j\}>$

En supposant $i \leq j$.

a-2 - La zone

Une zone (z) correspond à une image digitale de la mémoire d'images et définit un plan de celle-ci.

zone (z) ::= <tableau (n, m) pixels>

Dans notre cas n = m = 256.

a-3 - La partition

Une partition correspond à une "fenêtre" rectangulaire dans une zone de la mémoire d'images. Une partition est ainsi définie par les coordonnées de deux points opposés P_1 et P_2.

Partition (P_1, P_2) ::= <tableau (K, ℓ) pixels dans la zone z>

a-4 - Le pavé

Le pavé correspond à une partition particulière de 16 × 16 pixels

Pavé (x, y) ::= <tableau (16, 16) pixels>

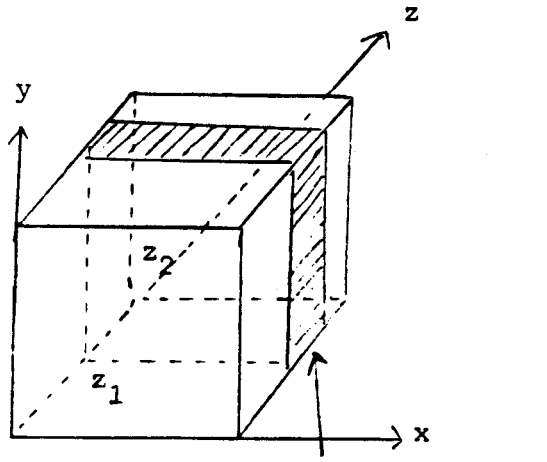
a-5 - Le pixel

L'objet le plus simple correspond au pixel accessible par le triplet <ℓ, c, z>

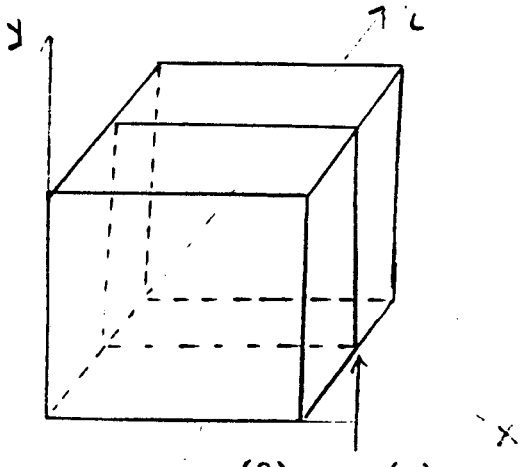
ℓ : numéro de ligne
c : numéro de colonne
z : numéro de zone

Pixel (x, y, z) ::= <$1/2^n$ couleur>

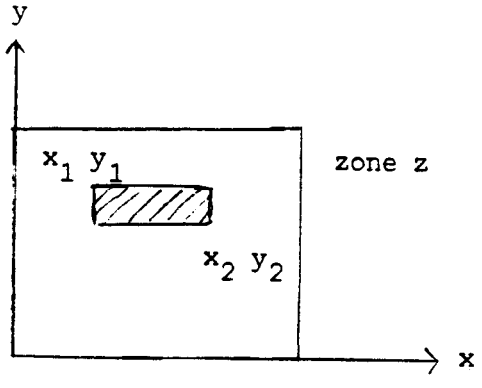
Dans notrecas n = 8, un pixel correspond alors à une couleur parmi 256.



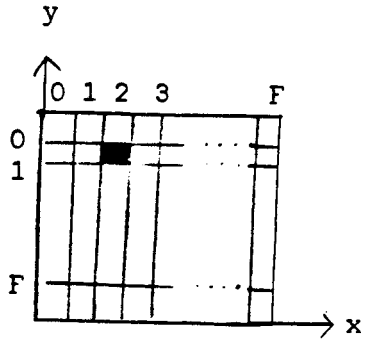
(1) Tranche (z_1, z_2)



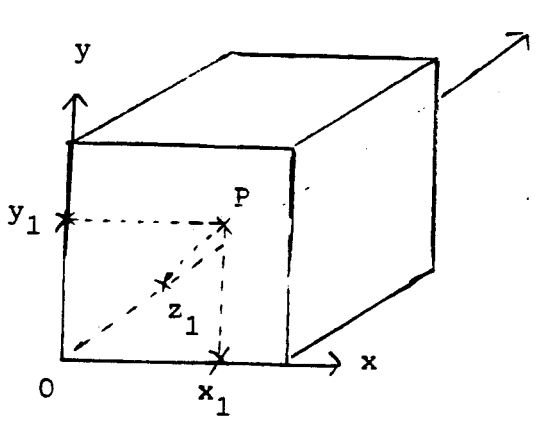
(2) zone (z)



(3) Partition (P_1, P_2)



(4) Pavé (1, 2)



(5) Pixel P(x_1, y_1, z_1)

Fig. IV.3. : Les objets de la mémoire d'images.

a-6 - La liste de valeurs

Cette notion permet l'introduction dans le système de listes de données correspondant à des paramètres divers. Un objet liste de valeurs se présente sous la forme générale suivante :

$$LV(P_1 ; P_2 ; P_3 ;...; P_n)$$

P_i étant un paramètre quelconque pouvant être : une couleur, un couple de coordonnées $[(x_1, y_1), (x_2, y_2)]$, une valeur immédiate (longueur...), un numéro de zone...

Un cas particulièrement intéressant est celui où P_i correspond à un triplet ((x, y), n, c) spécifiant qu'une suite de n pixels de valeur c commence au point de coordonnées (x, y) [représentation "Run-length" (GRA 83)]. Cette technique permet une compression dans le code de la liste.

b - Les objets concernant le convertisseur

Nous considérons les objets stockés dans les tables de couleurs. Là aussi, on peut définir diverses partitions de ces tables afin d'accéder à différents espaces d'adressages :

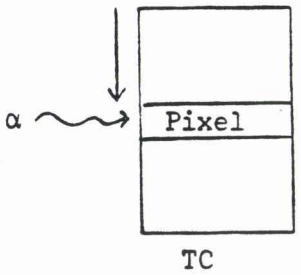
b-1 - Suite de 2^n emplacements : Chaque emplacement étant déterminé par son numéro :adresse dans la table (n : taille de la table).

TC ::= <2^n pixels >

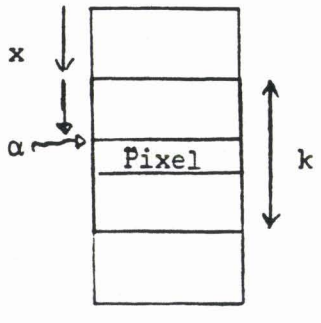
dans notre cas TC ::= <256 × 24 bits>

b-2 - Suite de K emplacements repérés grâce à un premier élément. L'accès à chaque élément dans la suite se fera relativement à ce premier élément.

Suite (x, k) ::= <k pixels>



(b-1)



(b-2)

Remarque : Pour représenter les objets nous avons pris des structures de données très simples (listes contigües), nous aurions pu prendre des listes chaînées, cependant la gestion de telles structures est plus complexe et n'apporte pas beaucoup d'améliorations par rapport à la représentation adoptée.

IV.2.1. Les macro-commandes graphiques

Les macro-commandes peuvent avoir des fonctions et rôles différents. Nous pouvons le classer en *fonction d'exécution* qui commandent l'exécution de séquences micro-programmées spécifiques, et de contrôle qui déterminent les débuts ou fins de séquence par exemple. Signalons les plus importantes à nos yeux :

IV.2.1.1. Les macro-commandes agissant au niveau de la mémoire d'images

a - Les macro-commandes de désignation

Ces macro-commandes permettent la désignation d'un objet mémoire :

- Désignation d'une zone.
- Désignation d'une partition.
- Désignation d'un pavé.
- Etc...

Cette fonction permet de réduire l'espace d'adressage aux coordonnées de l'objet désigné. (Et il y a donc une optimisation dans le codage de la macro-commande).

Cette désignation disparaît lors de la rencontre d'une autre macro-commande de désignation ou de fin.

Remarquons aussi, que de par leur codage même, certaines macro-commandes effectuent une désignation explicite ou implicite des objets sur lesquels elles vont opérer.

b - L'affectation

Cette macro-commande permet d'affecter à un objet O_1 la valeur d'un autre objet O_2.

L'objet O_2 pouvant être dans la mémoire d'images auquel cas un paramètre dans le code de la macro-commande le référence, ou une liste de valeur passée en paramètre. Le format général est le suivant :

MOVE O_1 ,O_2

O_2 : Référence à un objet en mémoire d'images ou liste de valeurs.

Une remarque s'impose quant à l'affectation d'un objet O_2 à un objet O_1, en effet, il existe un problème si les objets O_1 et O_2 ne sont pas de même type (Exemple O_1 : zone d'images et O_2 partition de taille différente).

Cette remarque est valable aussi pour les autres macro-commandes.

c - La combinaison d'objets

La combinaison d'images permet d'obtenir un objet à partir d'autres. On distingue généralement deux types de combinaisons :

- Le mélange.

- L'incrustation.

c-1 - Le mélange

Le mélange de deux objets O_1 et O_2 pour obtenir un objet O_3 peut se représenter ainsi :

$$O_3 := [(a \underline{op_1} O_1) \underline{op_2} (b \underline{op_3} O_2)] \underline{op_4} c$$

Les opérateurs op_i doivent avoir un sens pour les attributs des objets (exemples : et, ou, +, -,...).

a, b, c : valeurs déterminant les proportions de O_1 et O_2 pour obtenir O_3.

Exemple : La macro-commande : $O_3 := O_1 + O_2$ permet d'avoir l'effet de la superposition de deux images (deux transparents par exemple).

c-2 - L'incrustation

On parle aussi de masquage d'objets. La composition d'un objet O_3 par incrustation de deux objets O_1 et O_2 revient à affecter à O_3 la valeur de O_1 selon un critère donné ; celle de O_2 selon un second critère.

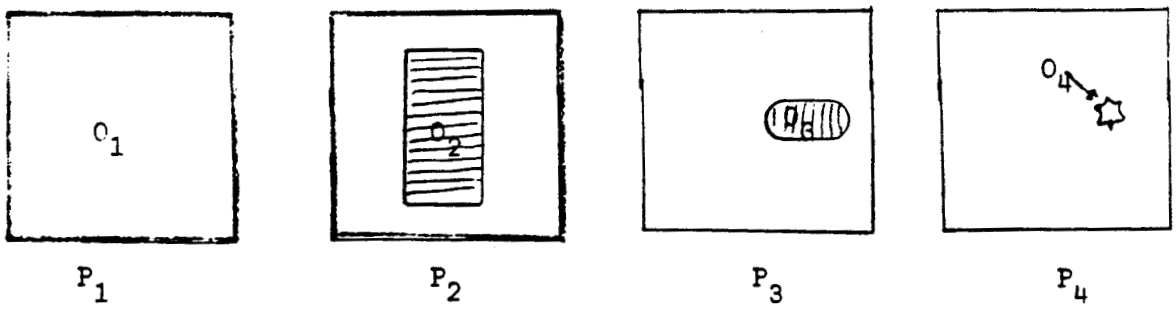
O_3 := si M alors O_1 sinon O_2 fsi

avec M ϵ {0, 1}.

On peut dès lors imaginer, par exemple, des macro-commandes à choix multiples :

O_1 := cas de M :
C_1 : O_j ;
C_2 : O_k ;
⋮
C_n : O_ℓ ;
fin cas

Une application de cette macro-commande est la détermination de surfaces cachées. Prenons l'exemple suivant :



Par combinaison de O_1, O_2, O_3 et O_4, on peut arriver à synthétiser des images diverses ; en attribuant des codes de profondeur aux pixels composant les O_i.

Soit, par exemple :

Profondeur	O_i
0	$\overline{O}_i$
1	O_1
2	O_4
3	O_3
4	O_2

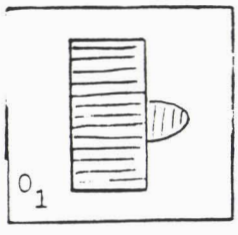
et par la macro-commande

O_1 := cas de profondeur

4 : O_2 ;
3 : O_3 ;
2 : O_4 ;
1 : O_1 ;

fin cas

On obtient



En supposant ces objets opaques.

d - Les transformations d'objets

Nous nous intéressons à des transformations géométriques dans le plan qui sont *locales* à chaque objet dans ce sens qu'elles n'interviennent que sur cet objet.

La géométrie considère cinq types de transformations ainsi que les paramètres associés :

- La rotation : centre et angle de rotation.
- La translation : vecteur de translation.
- L'homothétie : centre et rapport d'homothétie.
- la symétrie axiale : axe de symétrie.
- Le changement d'échelle : rapports de changement d'échelle, direction.

Ces transformations doivent être manipulées avec soin car il s'ensuit le plus souvent des problèmes de sous-échantillonnages dûs à la discrétisation de l'image mémoire. Un module de lissage permet la préparation à la visualisation de l'image ainsi obtenue.

e - Macro-commandes de production d'objets standards

Ces macro-commandes produisent des objets destinés à former une bibliothèque d'images standards qui servent dans la synthèse et l'animation. Ces objets peuvent être paramétrables ou non. Nous en donnons quelques exemples :

- Génération de formes géométriques standards

  * Cercle (Rayon, centre)
  * Droite ($x_1 : y_1$, $x_2 : y_2$)
  * Rectangle (L, ℓ)
  * Disque (Rayon, centre, couleur)
  * Grille de lignes (Epaisseur des lignes, espacement entre les lignes).
  * Etc...

- Motifs standards :

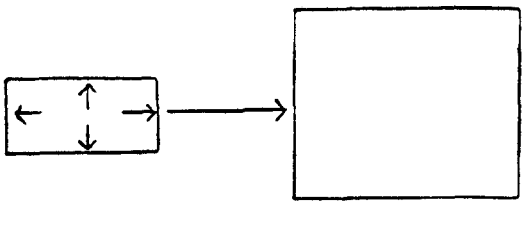
  * Génération de formes standards : maison, arbre,...
  * Génération de textures

    Etc...

La production de tels objets est assez importante en animation d'images. En effet, on peut obtenir un effet d'animation en visualisant une séquence représentant un objet dans des postures différentes et ordonnées. Nous remarquons, cependant, que la taille de la bibliothèque d'objets devient vite très importante ; un objet pouvant occuper un grand nombre de positions. Nous pouvons apporter quelques éléments de solution à ce problème :

- Action sur les paramètres si les objets sont paramétrables.

Exemple :



- Application de transformations géométriques ponctuelles (translation,...).

- Détermination d'un nombre fini de positions de l'objet pour faire de la "pseudo-animation". Ceci peut servir à certains domaines tels que : publicité,.

f - Le coloriage d'images

Le coloriage d'images est un processus très important en synthèse et en animation. D'une façon générale, il existe deux grandes familles d'algorithmes de coloriage (GRA 83) :

- Par utilisation de définitions géométriques de contour.

- Par travail direct dans l'image numérique.

Pour ce qui nous concerne, nous avons développé une macro-commande qui utilise un principe simple s'inspirant de cette deuxième famille :

COLORIER (O_i ; C_1, P_1 ; C_2, P_2 ;...; C_n, P_n)

où O_i : objet à colorier

C_i : couleur

P_i ::= NL, cd, cf

NL : numéro de ligne

cd : colonne où commence le coloriage

cf : colonne où il finit.

Le coloriage est fait ligne par ligne.

Un cas particulier de cette macro-commande consiste à affecter une même couleur à tous les points à colorier. Nous adoptons dans ce cas une syntaxe plus simple :

COLORIER (O_i ; C ; P_1 ; P_2 ;...; P_n)

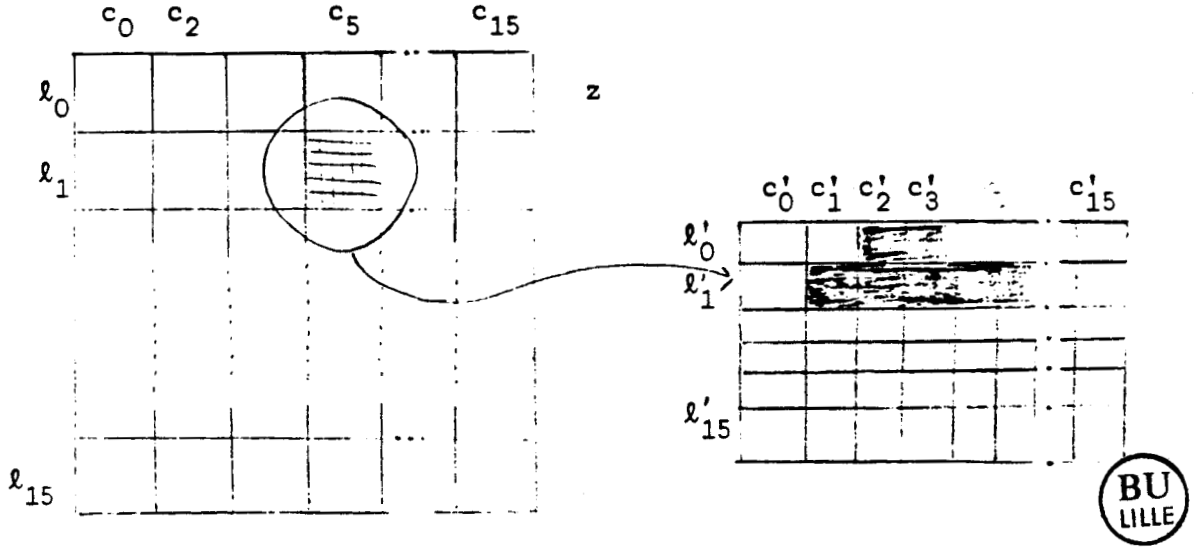
où O_i : objet à colorier.

C : couleur

P_i : même définition que précédemment.

Exemple : COLORIER PAVE ($P_{1,5}$; C ; 0, 2, 3 ; 1, 1, 5,...)
$P_{1,5}$ Pavé se trouvant à la ligne 1 et la colonne 5 de la zone z.

La zone z est supposée avoir été désignée par une macro-commande antérieure.



g - Macro-commandes diverses

Diverses autres macro-commandes peuvent être micro-programmées. Ce sont celles qui correspondent en pratique à une séquence de macro-commandes déjà existantes ; C'est le cas par exemple de la duplication n fois d'un objet O_i en O_{i+1}, O_{i+2},... O_{i+n}. Si on ne disposait que de la macro-commande MOVE O_1, O_2 l'algorithme serait le suivant :

```
Pour i := 1 à n
    faire
    | MOVE O(i+1),Oi
    fait
```

Ceci implique que le processeur hôte doit interpréter n fois la macro-commande MOVE ; ce qui est très pénalisant pour les performances du système. Une telle macro-commande gagnerait à être micro-programmée.

D'autre part, certaines macro-commandes correspondent à des cas particuliers d'autres et nécessitent un traitement moins complexe.

C'est le cas par exemple du tracé d'une droite horizontale ou d'un décalage vertical... On a donc intérêt à leur affecter des macro-commandes spécifiques.

De par l'architecture même de la machine ; les traitements les plus simples et les plus rapides sont obtenus pour des données ayant une capacité qui soit un multiple de 16 pixels (chaque processeur accédant au pixel qui se trouve dans sa fenêtre). On peut décomposer une donnée quelconque en une partition qui soit un multiple de 16 pixels et puis lui faire subir un changement d'échelle approprié. Cependant, certaines précautions doivent être prises dans un tel processus :

- Dans certains cas le traitement risque d'être plus complexe (cas d'un seul pixel par exemple).

- Le processus de décomposition ne doit pas être manié au niveau de l'interpréteur sinon il s'ensuit un temps d'overhead, mais au niveau de la macro-commande elle-même (décomposition au niveau de P_M qui va en fait déclencher plusieurs séquences).

II.2.2.2. Commandes agissant au niveau du convertisseur

Nous distinguerons deux types de commandes agissant au niveau du convertisseur en considérant celui-ci connecté au processeur hôte et géré par ce dernier.

a - Commande de visualisation d'un plan de l'image qui consiste à émettre vers l'A.D.M. le numéro de zone à afficher sur l'écran.

b - Commande de chargement des tables de couleur

Cette commande correspond à la macro-commande d'affectation en mémoire d'images.

Chargement $TC(L_1, C_1 ; L_2, C_2 ;\ldots; L_n, C_n)$

correspond au chargement de la table qui est affectée au processeur à partir de la position 0.

Chargement TC(x ; L_1, C_1 ; L_2, C_2 ;...; L_k, C_k)

chargement des m éléments de la TC à partir de la position x. Ce qui correspond à un cas plus général.

L_i : longueur = nombre d'emplacements.

C_i : couleur qui sera chargée dans ces emplacements.

IV.3. Réalisation des macro-commandes

Nous pouvons distinguer plusieurs modes d'adressage de la mémoire d'images selon le traitement requis par la macro-commande :

- L'adressage est confié au processeur maître, les processeurs de traitement exécutent des micro-programmes sur les pixels se trouvant dans leurs fenêtres respectives (Exemples : Affectation d'une valeur à une zone, tracé d'une droite horizontale, addition de zones,...).

- Une partie de l'adressage seulement est confiée au processeur maître, les processeurs de traitement délivrent un déplacement relatif à cette partie de l'adresse. (Dessins de courbes quelconques dans une zone,...).

- L'adressage est confié exclusivement aux processeurs de traitement.

Dans chacun des cas le contrôle de lecture/écriture est confié à l'un ou l'autre des partenaires (P_M, Processeurs élémentaires).

IV.3.1. Les micro-programmes parallèles

L'exécution d'une macro-commande correspond à l'activation parallèle d'un ensemble de micro-programmes spécifiques par le processeur maître. Ceci se faisant en fonction du code de la macro-commande qui détermine ainsi sa fonction et de ses paramètres déterminant des consignes diffusées aux différents micro-programmes.

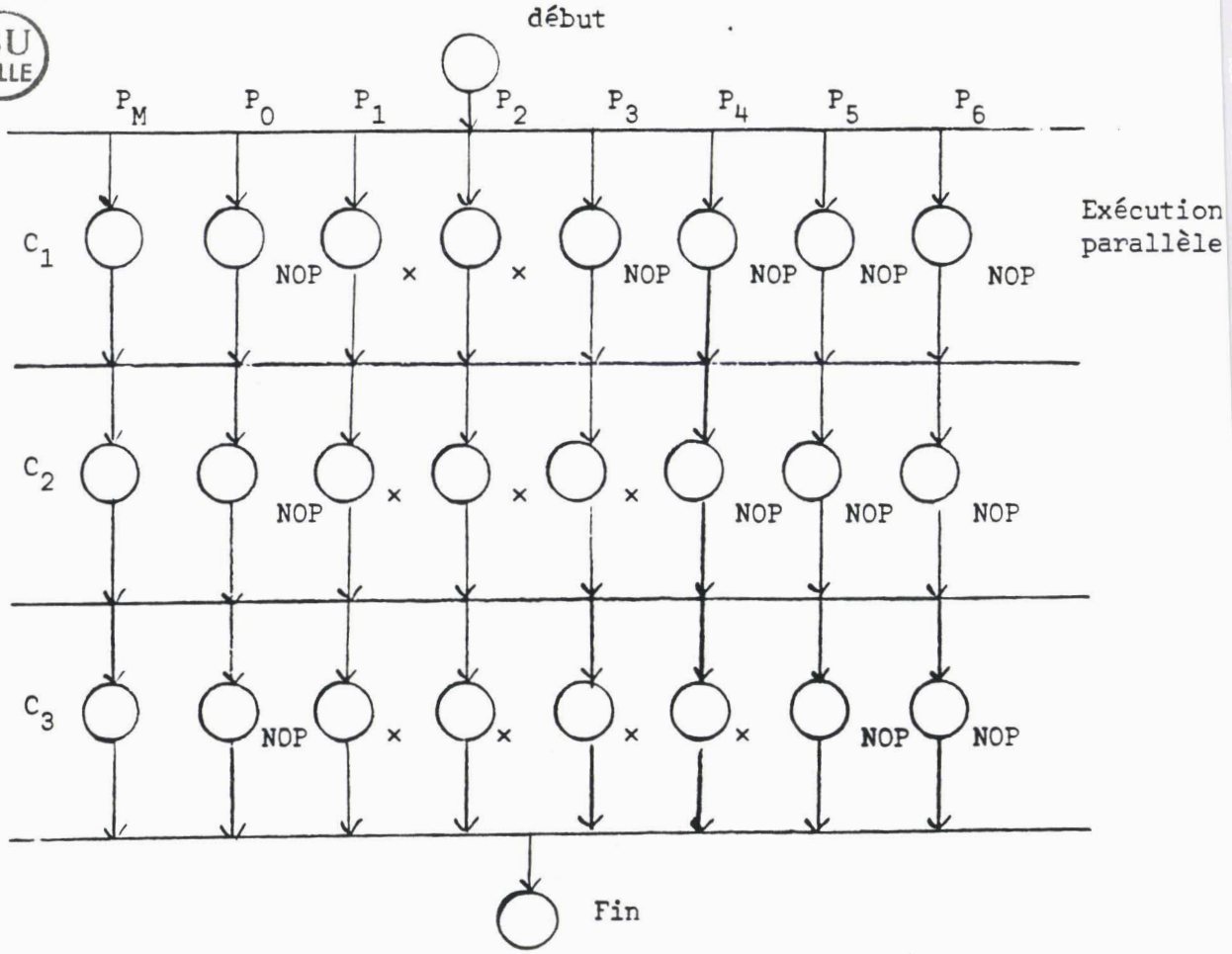
Dans la mise en oeuvre présente l'adressage ainsi que le contrôle en mémoire d'images sont confiés au processeur maître, deux raisons principales ont conduit à ce choix :

- La première raison est d'ordre matériel. Munir chaque processeur de sa propre voie d'adressage est assez délicat à réaliser pratiquement vu le nombre de connexions qu'il faut avoir ainsi que les coûts des circuits associés.

- D'un autre côté ceci favorise des traitements de type SIMD étendu à un certain nombre de processeurs du réseau comme nous allons le voir sur l'exemple suivant

Exemple : Soit la macro-commande suivante :

	P_0	P_1	P_2	P_3	P_4	P_5	P_6
C_0							
C_1		*	*				
C_2		*	*	*			
C_3		*	*	*	*		
C_4							
C_5							



Les traitements marqués par × correspondent à la même micro-instruction s'exécutant sur des processeurs différents et des données différentes (SIMD).

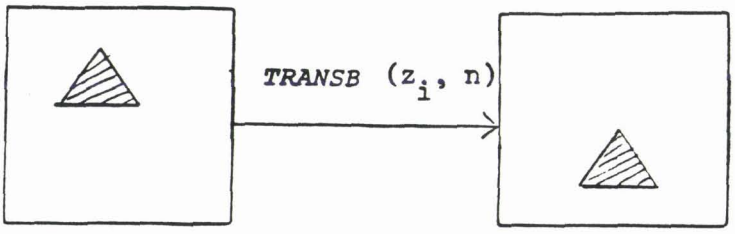
C_i : colonne numéro i.

Dans l'exemple suivant nous allons donner l'algorithme parallèle d'une macro-commande simple consistant en une translation d'une zone mémoire (z_i) (soit 256 × 256 pixels) de n lignes vers le bas. Nous adoptons arbitrairement le formalisme suivant :

M(ℓ, c, z) : désigne un pixel en mémoire d'images dont l'adresse correspond au triplet <ℓ, c, z> numéro de ligne, numéro de colonne et numéro de zone respectivement. ℓ, c, z sont des identificateurs d'adresses contrôlés par le processeur maître.

Nous désignerons par $X la séquence X se déroulant en mode SIMD sur les différents processeurs du réseau. Le reste de la macro-commande étant exécuté par le maître • a et b désignent des variables de travail.

```
TRANS B (z_i, n)
 début déclare a, b variables pixel
|     z := z_i ;
|   Pour i := 1 à n faire
|       ℓ := 0
|   Pour c := 0 à 15 faire
|       ℓ := 0 ;
|       $a := M(ℓ, c, z) ;
|   Pour ℓ := 1 à 255 faire
|       $b := M(ℓ, c, z) ;
|       $M(ℓ, c, z) := a ;
|       $a := b ;
|   fait
|  fait
| fait
 fin
```



Nous donnerons en annexe la liste ainsi que les fonctions des macro-commandes développées ou en cours de développement.

II.3.2. *Interprétation et exécution des macro-commandes*

Les macro-commandes sont des suites de codes émis par le processeur hôte au processeur maître selon un principe de producteur-consommateur (cf. III.2). Ces codes spécifient la macro-commande et les paramètres sur lesquels elle porte selon le format suivant :

$$\text{codop } P_1, P_2, \ldots, P_n$$

codop : code opération spécifiant la fonction de la macro-commande.

P_i : paramètre (adresse, couleur,...).

Le processeur maître doit interpréter ces codes et puis diffuser des commandes vers les processeurs du réseau dans le but d'exécuter la fonction désirée. Nous avons prévu à cet effet un noyau qui se compose des quatres modules suivants :

- Le module d'initialisation.
- Le module de synchronisation.
- L'analyseur de macro-commandes.
- Le module de lancement des séquences.

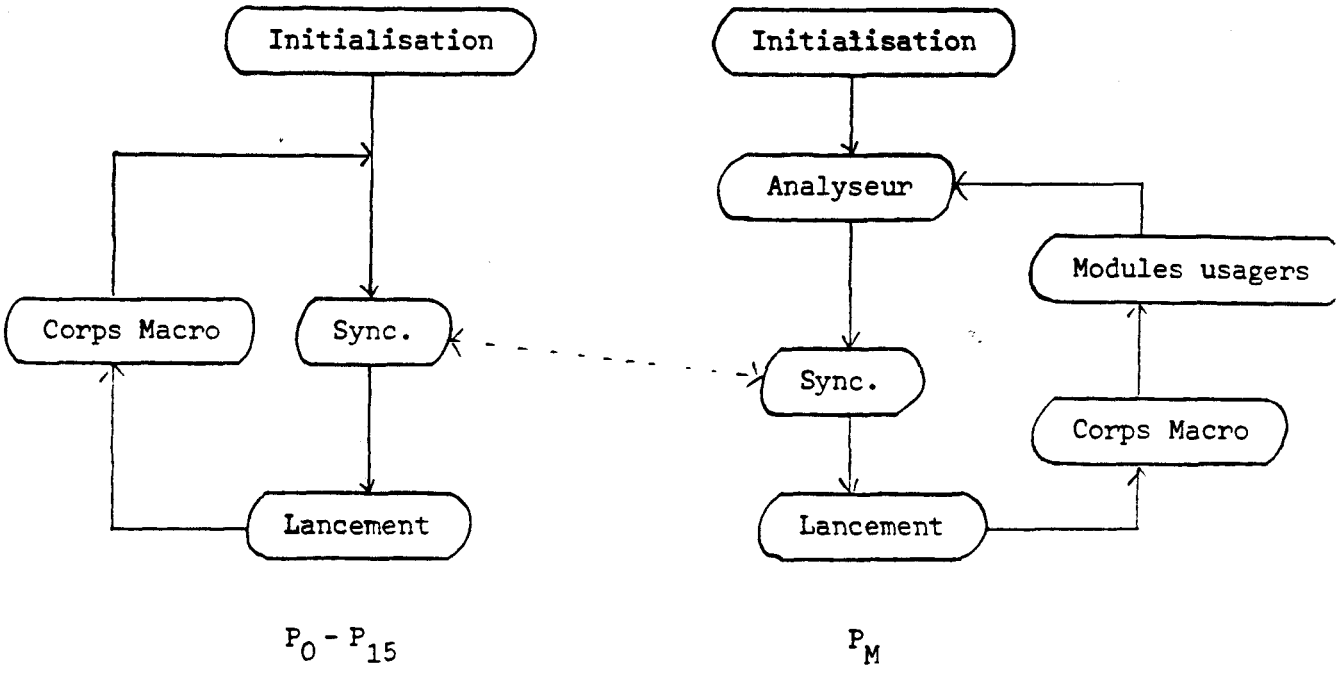


Fig. IV.4. : Principe du noyau.

IV.3.2.1. Le module d'initialisation

Ce module assure l'initialisation des différents organes de la machine :

- Initialisation des indicateurs de synchronisation.

- Initialisation des registres de communication et de routage.

- Initialisation de la bascule d'état B assurant la synchronisation avec le 8085 (P_M seul).

- Initialisation des commandes contrôlant la mémoire de données (adresse, bit de lecture/écriture).

Ce module est implanté à l'adresse ooo des mémoires de micro-programmes de telle sorte qu'une commande RESET permette son exécution.

Ce module se termine par un branchement à l'analyseur dans le cas du processeur maître et au module de synchronisation dans le cas des processeurs de traitement.

IV.3.2.2. L'analyseur

Son rôle est de :

- Recevoir une suite de codes spécifiant la macro-commande émise par le processeur 8085 à travers une voie parallèle 8 bits selon un principe de producteur-consommateur ; le producteur étant le 8085, le consommateur P_M (voir chapitre III).

- Interpréter ces codes qui correspondent au code de la macro-commande à exécuter ainsi qu'aux paramètres à lui fournir.

- Diffuser tout ou partie de ces codes vers le réseau de processeurs.

Ce module se termine par un branchement au module de synchronisation.

IV.3.2.3. Le module de synchronisation

Les différentes séquences rédigées par l'usager peuvent se terminer à des instants différents (activités locales entraînant des boucles de longueur indéterminées...). L'enchaînement des commandes ne doit se faire que lorsque tous les processeurs sont prêts à recevoir les prochains codes de macro-command Ceci implique l'existence d'un module de synchronisation étendu à ces processeu mettant en oeuvre les outils matériels de rendez-vous développé au Chapitre II.

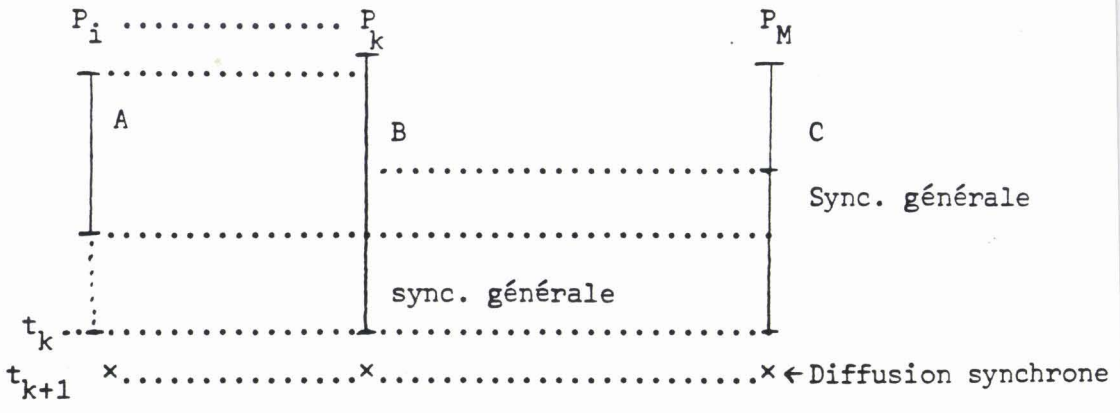


Fig. IV.5. : Principe du module de synchronisation.

Nous voyons dans la Fig. IV.5. le principe de fonctionnement. Les séquences micro-programmées A, B et C peuvent avoir une longueur quelconque. Le synchronisme est garanti et à l'instant t_{k+1} tous les processeurs démarrent des micro-instructions synchrones assurant la diffusion des consignes depuis P_M.

IV.3.2.4. Le module de lancement

Ce module permet le branchement à la séquence concernée grâce aux codes spécifiant la macro-commande acquis par le module d'analyse et diffusés vers l'ensemble des processeurs du réseau.

Une difficulté surgit quant à la mise en oeuvre de cette technique sur le 8 × 300 étant donnée l'absence sur ce micro-processeur d'instruction d'appel de sous-programmes. Cette difficulté a été contournée par l'utilisation d'une table de sauts indexés.

Le code de la macro-commande correspond à une entrée dans la table des sauts. Chaque séquence correspondant au corps de la macro-commande se termine par un retour au module de synchronisation.

IV.3.2.5. Modélisation à l'aide de réseaux de Pétri

Nous proposons dans ce qui suit une modélisation du système à l'aide des réseaux de Pétri. Nous employons les abréviations suivantes :

Init. : Module d'initialisation.
Sync. : Module de synchronisation.
S(i) : Emission du signal d'arrivée au rendez-vous.
Lancement : Module de lancement.
M_i, M'_j : Séquences micro-programmées spécifiques de traitement des macro-commandes

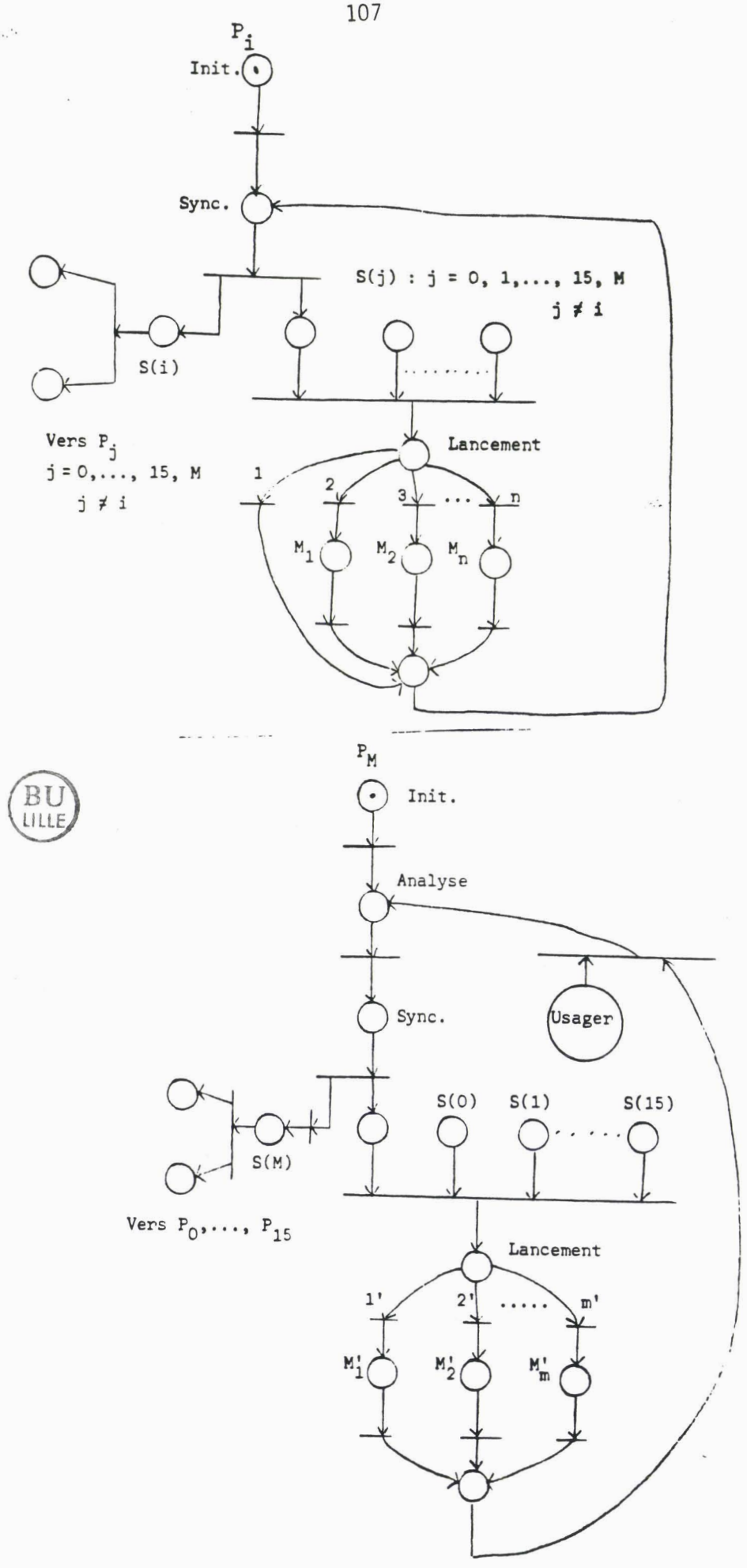


Fig. IV.6. : Modélisation à l'aide de réseaux de Pétri.

II.3.3. Réalisation matérielle

L'analyseur actuel est conçu pour traiter des macro-commandes sur deux ou trois octets. Il est simple de le modifier pour qu'il accepte d'autres formats.

- Octet 1 : Définit le code opération de la macro-commande. Si le bit 2^7 de cet octet est à 1 cela signifie que la macro-commande comporte 3 octets, sinon elle n'en comporte que deux. L'analyseur stocke cet octet dans un registre interne et le diffuse via le bus de communication RCOM à l'ensemble des processeurs du réseau qui le stockent à leur tour dans des registres internes. Cet octet servira d'index au module de lancement.

- Octet 2 : Définit les modalités d'adressage. Dans notre application l'adressage est confié exclusivement au processeur maître. Cet octet n'est donc pas transmis aux autres processeurs du réseau. il sert à définir :
  - Des zones de mémoire.
  - Des modes et pas de balayage.
  - Des valeurs limites de zones.

Le processeur maître se charge aussi du contrôle de l'accès en mémoire d'images.

- Octet 3 : Il peut contenir un paramètre quelconque et il est de ce fait communiqué à tous les processeurs du réseau. Dans le cadre des traitements graphiques il peut être :
  - Une valeur (couleur,...).
  - Une indication sur l'adressage.
  - Etc...

IV.4. Le noyau de logiciel

Le noyau constitue la couche logicielle permettant à l'usager l'utilisation du système sans avoir le souci des contraintes d'enchaînement et de synchronisation. Il doit donc fournir les fonctions suivantes :

- Chargement des séquences micro-programmées en mémoire.
- Exécution de ces séquences.
- Analyse, interprétation et exécution des macro-commandes.

Diverses autres fonctions telles que : Edition de textes, compilation, gestions de fichiers sont assurés par le logiciel hôte (cf. Chapitre III).

IV.4.1. Le chargeur

Le but du chargeur est d'implanter des séquences de micro-instructions dans les différentes mémoires de micro-programmes relatives aux différents processeurs du réseau. A cet effet, nous avons prévu deux programmes de chargement C8251 et LOADER.

a - Le chargement intéractif : Programme C8251

Ce programme permet le chargement d'un micro-code 8 × 300 tapé au clavier dans un ou plusieurs processeurs du réseau (désignés explicitement au clavier) à l'adresse désirée.

Ceci permet de dupliquer les mêmes micro-instructions dans des processeur différents et d'assurer ainsi un fonctionnement SIMD pour ces processeurs.

Il est aussi possible de provoquer l'exécution d'un ensemble de micro-instructions sous le contrôle d'une horloge automatique ou sous celui du processeur trace permettant ainsi la mise au point des micro-programmes.

De plus, un micro-programme écrit peut être sauvegardé sur disquette pour une exploitation ultérieure.

b - Le chargement automatique : Programme LOADER

Ce programme exploite des fichiers de 8 K enregistrements de micro-instructions à charger dans les mémoires de micro-programmes des différents processeurs. Ces fichiers sont obtenus soit par l'intermédiaire du CROSS-ASSEMBLEUR 8 × 300 soit par codage direct.

Le chargement se fait sous LOADER par l'assignation d'un fichier à un ou plusieurs processeurs.

IV.4.2. Logiciel de gestion de macro-commandes

Deux programmes permettant d'éditer, de cataloguer éventuellement et d'exécuter les macro-commandes ont été mis au point ; ce sont les programmes ADM et IMAGE, le premier permet l'exécution intéractive des macro-commandes tapées au clavier et ne permet pas de garder une trace. Le second permet de concevoir des séquences de macro-commandes que l'on peut sauvegarder dans des fichiers pour des exploitations ultérieures. Une séquence de macro-commandes est lancée explicitement au clavier par l'utilisateur afin d'être exécutée. Un compteur de boucle permet l'exécution d'une séquence plusieurs fois. Ainsi une image complexe peut être obtenue par combinaison d'images plus simples.

IV.5. Conclusion

Nous avons développé dans ce chapitre les principes de base mis en oeuvre pour fournir un noyau de logiciel permettant l'utilisation adéquate de la machine. Cependant, à ce niveau les macro-commandes sont des structures figées constituées de codes de bas niveaux (codes héxadécimaux), de même, pratiquement aucune structure de contrôle n'existe pour assurer la formation de programmes utilisateur.

Le chapitre suivant décrira la structure d'un interpréteur réalisé pour pallier à ces inconvénients.

SOMMAIRE DU CHAPITRE V

V. L'INTERPRETEUR

V.1. Introduction

L'interpréteur correspond au niveau le plus élevé du système. Son rôle est de produire et de gérer de façon intéractive ou non des séquences de macro-commandes.

Les points suivants déterminent le cahier de charges de l'interpréteur :

- Les macro-commandes correspondent à des séquences de codes de bas niveau. L'interpréteur permet d'offrir des structures de données et de contrôles assurant leur identification et leur manipulation des macro-commandes et la formation ainsi de programmes utilisateurs.

- Il permet aussi de décharger l'utilisateur de tâches logicielles utilitaires telles que gestion de ressources (mémoire, fichiers,...) et communication (interface avec le processeur maître,...).

D'une façon générale, l'interpréteur permet d'offrir des outils permettant une programmation de haut niveau en rendant transparente l'architecture du système. Ceci nous a amené à implémenter un langage évolué.

Les critères principaux du choix d'un tel langage sont les suivants :

- Intéractivité.
- Souplesse.
- Extensibilité.
- Aide à la mise au point dynamique.

Ces critères favorisent beaucoup plus les langages interprétés que les langages compilés.

Notre choix s'est porté sur le langage FORTH ; plusieurs points ont motivé ce choix, parmi eux ceux cités ci-dessus d'une part. D'autre part, FORTH présente les caractéristiques générales suivantes :

- Faible capacité du noyau ; ceci le rend facilement implémentable sur le système hôte.

- Possibilité de fonctionnement en mode compilation ou interprétation ce qui le rend apte à divers types d'exploitation.

- Beaucoup plus qu'un langage, FORTH présente les caractéristiques d'un système d'exploitation et permet donc d'intégrer les fonctions de ce dernier ; ceci est très important dans le domaine de l'intéractivité en rendant les temps d'overhead négligeables.

- La structure du langage est très simple, la syntaxe est non contraignante ce qui rend facile son utilisation.

- Il est très fréquent en synthèse et en animation d'images de vouloir produire une image qui soit obtenue par combinaison d'autres plus simples (voir Chapitre IV). Ce procédé doit être simple et implique l'existence de liens de chaînage entre ces images. De par sa structure même, le langage FORTH (langage chaîné) offre directement ces mécanismes.

FORTH a été normalisé par l'international FORTH Standard Team (STT 83).

V.2. Présentation du langage FORTH

Nous présentons dans ce paragraphe, les concepts de base de ce langage qui nous apparaissent assez importants, le lecteur intéressé peut consulter (BRO 81, STT 83, LOE 81) la liste n'est pas exhaustive.

V.2.1. Concepts de base

Le langage est bâti autour de trois concepts de base qui sont :

- Le mot.
- Le dictionnaire.
- Les piles : - La pile de données.
  - La pile de retour.

V.2.1.1. Le concept de mot

Un programme FORTH est une suite de mots, l'utilisateur peut donner un nom à une suite quelconque de mots, ce nom deviendra alors un nouveau mot et ainsi de suite.

Il existe deux sortes de mots :

- Les mots "de base" qui sont des suites de codes machine et qui constituent le noyau du langage.

- Les mots secondaires qui sont des mots crées à partir des mots de base ainsi que d'autres mots secondaires.

En FORTH la création d'un nouveau mot se fait selon le format suivant :

```
: NMOT  VMOT1  VMOT2 .... VMOTn ;
```

où NMOT : nom du nouveau mot.
$NMOT_i$: mot existant déjà.

: mot du langage indiquant le mode compilation (création d'un nouveau mot
; Fin de création.

Donc à chaque définition d'un nouveau mot il y a chaînage vers les définitions des mots composants ce mot et existant déjà.

Exemple :

```
: N1  A  C ;
: N2  A  B  D ;
: N3  N1  N2  A ;
```

où A, B, C, D sont des mots du langage prédéfinis, à la fin de chaque compilation les nouveaux mots sont crées et peuvent être accédés.

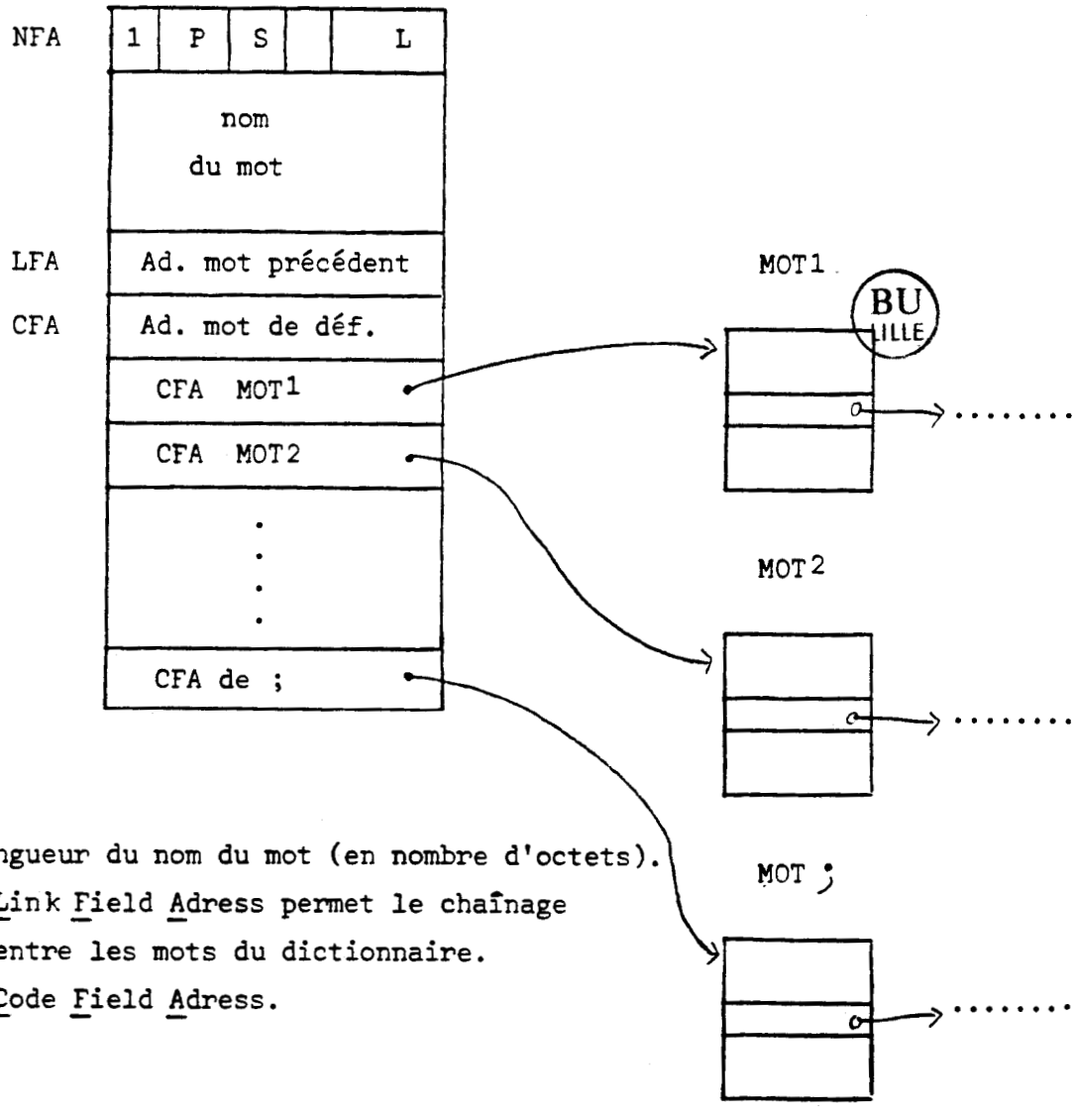
V.2.1.2. Le concept de dictionnaire

Le dictionnaire regroupe tous les mots connus du langage, la création d'un nouveau mot est possible par son intégration dans ce dictionnaire.

Cet élément occupe une taille importante de la mémoire.

Soit le mot défini de la façon suivante :

: nom du mot MOT1 MOT2 ..... MOTn ;



L : longueur du nom du mot (en nombre d'octets).
LFA : Link Field Adress permet le chaînage entre les mots du dictionnaire.
CFA : Code Field Adress.

Le CFA peut contenir trois sortes d'informations :

- Soit du code exécutable.
- Soit des paramètres.
- Soit enfin les adresses des mots qui le constituent.

NFA : Name Field Adress : zone définissant le nom du mot.

La recherche dans le dictionnaire se fait par l'intermédiaire des LFA.

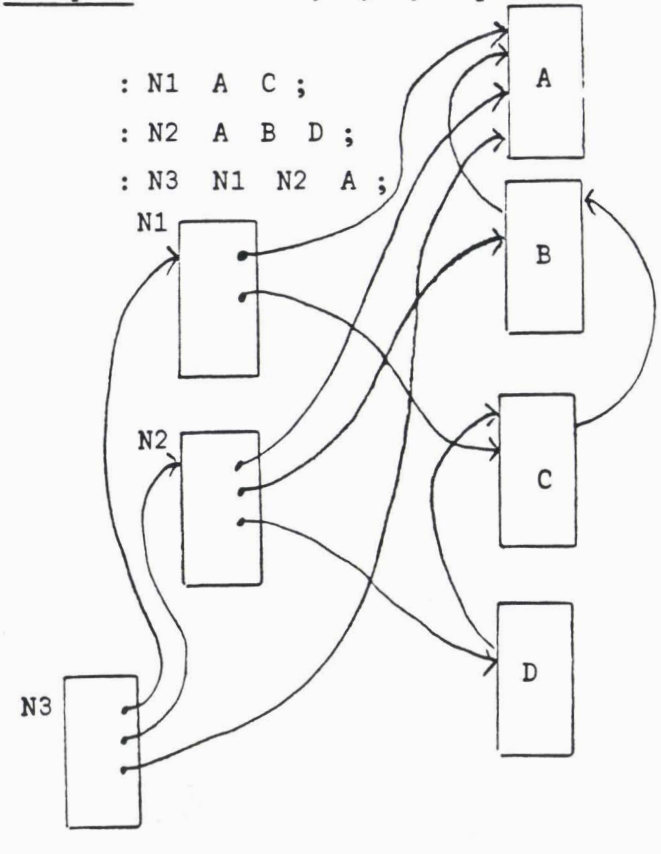
Le bit S (SMUDGE) permet de valider une définition dans le dictionnaire et permet, par exemple, d'y incorporer des codes machines ainsi que l'exécution de mots récursifs.

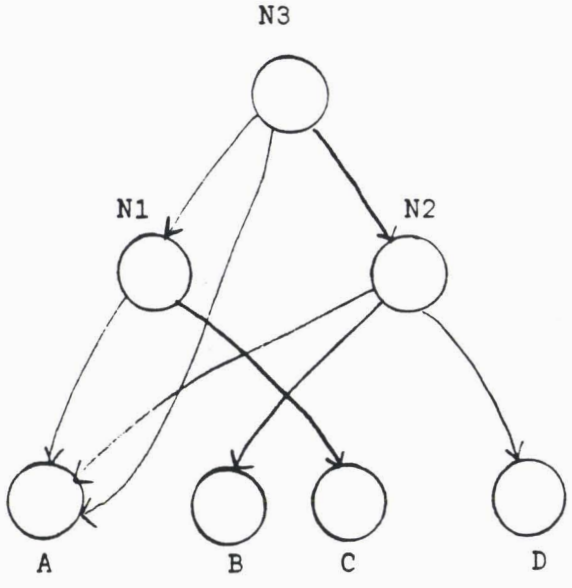
Le bit P (Precedence bit) permet une exécution *immédiate* du mot concerné à cet effet, un mot FORTH (IMMEDIATE) permet de forcer ce bit à 1.

Le MSB du NFA étant à "1" ceci facilite la recherche dans le dictionnaire des blocs correspondant à chaque mot.

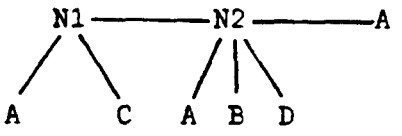
La définition d'un mot crée donc une arborescence dont les feuilles sont obligatoirement des mots de base (code machine).

Exemples : Soient A, B, C, D quatre mots de base. Et les définitions suivantes :





En fait, il existe un ordre d'exécution. Par exemple, pour N3 :



L'exécution se fait de la gauche vers la droite.

Pour ce qui nous concerne, l'application de ce principe est directe. En effet, soit M = $\{M_1,\ldots, M_n\}$ l'ensemble des macro-commandes telles qu'elles sont définies au Chapitre IV. Une première étape consiste à intégrer ces macro-commandes dans des mots FORTH de la façon suivante :

: M_1 MOT 1 MOT 2 ..... MOT I ;

: M_2 MOT'1 ..... MOT'J ;

.
.
.
.

: M_n $MOT^{(n)}1$ ..... $MOT^{(n)}L$;

$MOT^{(K)}i$: mot FORTH précédemment défini.

Une image quelconque composée à partir de ces macro-commandes correspondra à :

: IMAGE a_1 M_1 a_2 M_2 ..... a_k M_k a_{k+1} ;

où les a_i sont des mots FORTH servant de structures de contrôle.

Le mot image sera alors inséré dans le dictionnaire et pourra servir à la compilation d'autres mots ou pour être exécuté.

La séquence $\{a_i\ M_i\}$ peut aussi être exécutée directement en mode exécution.

V.2.1.3. Les piles

a - La pile de données

L'exécution des opérations arithmétiques ou logiques en FORTH se fait par l'intermédiaire d'une pile dite de données ; à cet effet, ces informations sont présentées en notations polonaises post-fixées.

Des mots FORTH permettent l'exploitation de cette pile ce sont par exemp

- DUP : duplique le sommet de pile.
- DROP : détruit le sommet de pile.
- SWAP : échange les valeurs du sommet et du sous-sommet de pile.
- ROT : permet une permutation circulaire sur les trois dernières données entré

b - La pile de retour

Cette pile sert à mémoriser les adresses de retour aux mots appelants.

FORTH donne l'accès à cette pile par l'intermédiaire des deux mots suivants :

> R : transfère la cellule du sommet de la pile des données vers le sommet de la pile de retour.

R > : effectue l'opération inverse.

V.2.2. Manipulations diverses

V.2.2.1. Les mots arithmétiques et logiques

FORTH manipule les nombres suivants :

- Les nombres en simple longueur (16 bits) signés ou non.

- Les nombres double longueur (32 bits) signés ou non.

- Les nombres flottants.

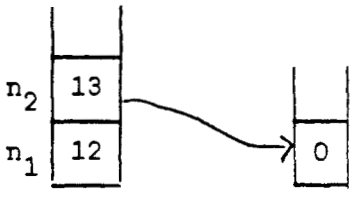
Les opérations effectuées sont les opérations arithmétiques et logiques usuelles (+, -, *, et, ou, >, <, =,...).

Leur emploi utilise la pile de données.

Les opérations s'effectuent entre les valeurs situées au sommet et au sous-sommet de la pile. Le résultat est stocké en sommet de pile (les valeurs initiales sont perdues).

Exemple :

12 13 = (retour)



Le mot = met 0 si $(n_1) \neq (n_2)$ 1 sinon.

Et permet de tester l'égalité de deux valeurs.

V.2.2.2. Les structures de contrôle

Ce sont là les structures de contrôle classiques :

- Alternatives : IF (vrai) ELSE (faux) ENDIF

- REPETITIVES : DO.....LOOP...

- BOUCLES INFINIES : BEGIN ... AGAIN

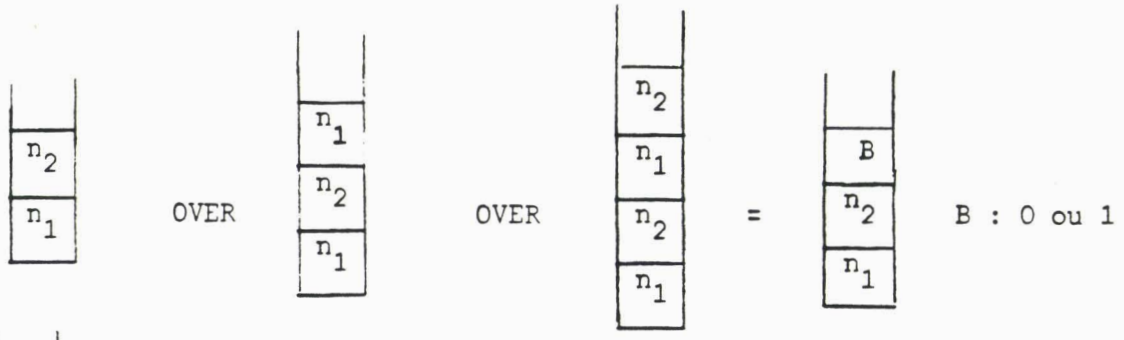
- etc...

Exemple :

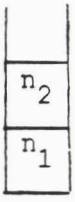
```
: EXEMPLE OVER OVER = IF  "nombres égaux "ELSE + ENDIF ;
```

Dans cet exemple à but purement académique, deux nombres n_1 et n_2 sont supposés être dans la pile ; on les recopie puis on effectue l'addition s'ils sc différents sinon on émet le message "nombres égaux".

Le mot OVER permet de dupliquer le sous-sommet de pile au sommet de celle-ci :



IF



.....

V.2.2.3. Les structures de données

FORTH permet l'implantation de structures de données statiques (tableaux, et dynamiques (LISTES, FILES,...) grâce à des mécanismes de gestion de la mémoire centrale. D'autres mécanismes de gestion de la mémoire secondaire permettent la manipulation d'ensembles plus importants d'informations.

V.2.3. Conclusion

Diverses autres fonctions de FORTH n'ont pas été abordées ici. L'utilisateur peut commencer à travailler avec un vocabulaire très réduit et peut le compléter au fur et à mesure par de nouveaux mots secondaires ou de base (FORTH permet l'accès au code machine).

De plus, l'utilisateur peut créer d'autres vocabulaires correspondants à des contextes différents et des applications différentes. L'accès à tel ou tel vocabulaire est possible directement. Leur organisation est identique à celle des mots dans le dictionnaire.

Les vocabulaires de base sont les suivants :

- FORTH : regroupe les mots FORTH.

- ASSEMBLER : regroupe tous les mots permettant la programmation en assembleur.

- EDITOR : mots servant à l'édition : modification de la chaîne de caractéres composant un mot, visualisation de la structure d'un mot, recherche d'un mot...

Notre objectif est de spécifier un vocabulaire pour notre application dont le but est d'interpréter des commandes de l'utilisateur et de produire des codes spécifiant des macro-commandes. D'autre part il est possible de créer des séquences de macro-commandes cataloguées et de produire ainsi une bibliothèque de mots correspondant à ces séquences (séquences animées, jeux vidéos,...).

V.4. L'interpréteur de macro-commandes

Cet interpréteur correspond aux modules suivants :

- Noyau FORTH.

- Ensemble de mots gérant les macro-commandes de base.

- Ensemble de mots représentant des séquences de macro-commandes.

Une macro-commande est identifiée par un nom (qui correspond à un mot FORTH) ainsi qu'un compteur de paramètres spécifiant le nombre de paramètres requis par cette macro-commande. Les codes correspondant à chaque macro-commande sont empilés dans la pile de données, un mot FORTH permet d'émettre ces différents codes vers le processeur maître du système multiprocesseur.

V.4.1. Fonctionnement du système

Les mots de gestion et de production des macro-commandes sont les suivants :

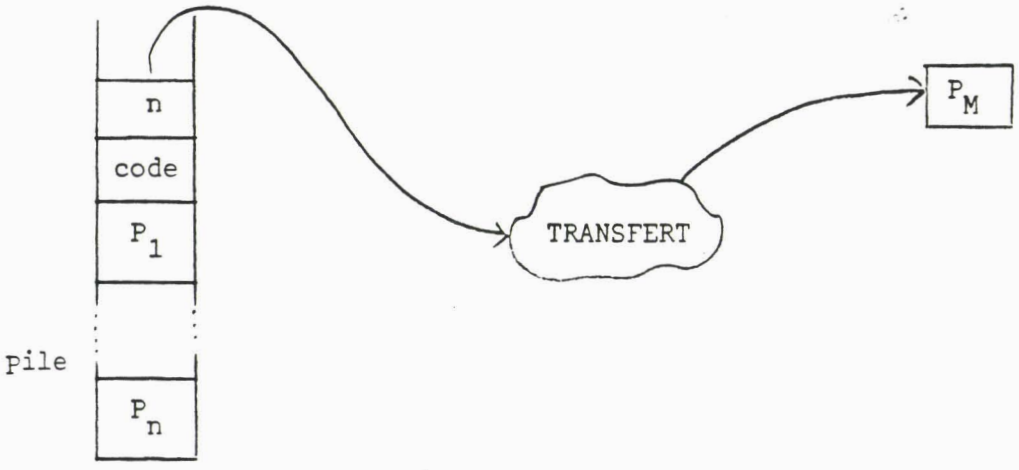
- Mots définissant les macro-commandes; ces mots placent le code correspondant à la macro-commande dans le sous-sommet de pile, le nombre d'octets dans le sommet de pile.

- Mot gérant l'interface avec le processeur maître. Ce mot transfère les octets spécifiant la macro-commande selon le schéma suivant :

* Emission du code de la macro-commande

* Emission des paramètres (selon la valeur du compteur).

Ces paramètres sont eux aussi transférés à partir de la pile ceci suppose donc qu'un prétraitement les y a empilés.



Module TRANSFERT

```
n := (SP) ; dépiler        co SP : Sommet de pile fco
tant que n ≠ 0 faire
    | TRANSFERER (SP) ;
    | dépiler ;
    | n := n-1 ;
  fait
fin
```

MACRO-COMMANDE : MC

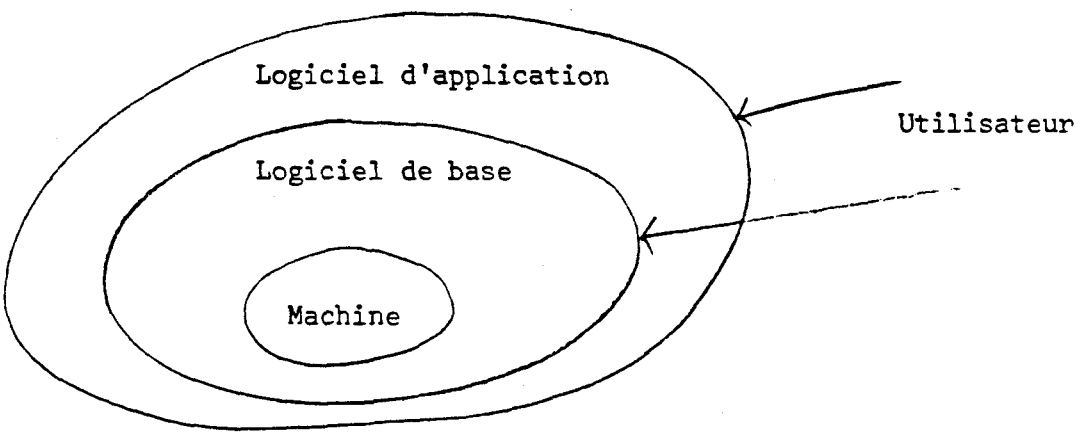
```
début
  | empiler (code macro) ;
  | empiler (nombre d'octets) ;
  | TRANSFERT ;
fin
```

Transférer (x) : permet le transfert de la valeur x depuis le 8085 vers P_M.

V.4.2. Les programmes de l'utilisateur

Nous avons développé dans les chapitres IV et V le logiciel de base dont la tâche est de gérer la machine et d'offrir ainsi à l'utilisateur une "vue" simple et non contraignante de celle-ci.

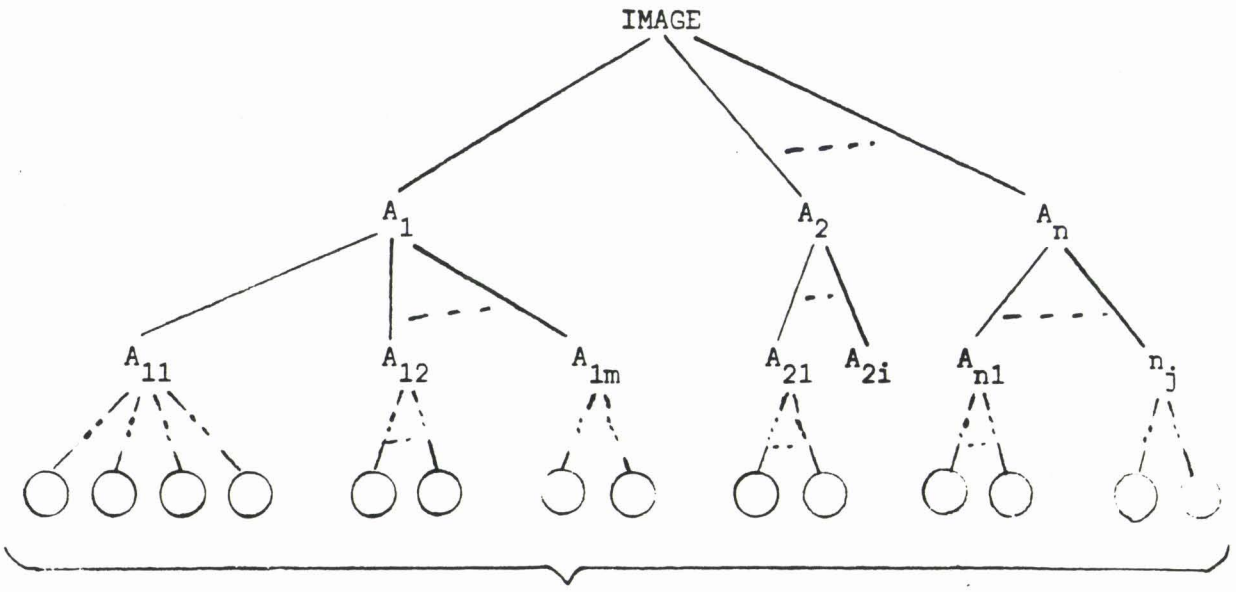
Une couche supplémentaire s'avère généralement nécessaire. Composée de programmes exécutants des fonctions standards (fonctions de calcul, de synthèse d'images standards, de tests,...), cette couche constitue le logiciel d'application qui exploite la machine à travers le logiciel de base.



Grâce à cet interpréteur l'utilisateur peut écrire des programmes de synthèse et de manipulation d'images. Une fois écrit et mis au point un tel programme peut être catalogué sous un nom, et devenir ainsi un mot du langage FORTH. On peut ainsi créer d'autres mots composés d'un ensemble de programmes, etc...

Une telle structure permet donc une grande modularité qui est un élément important pour la conception de programmes structurés, ainsi qu'une souplesse appréciable.

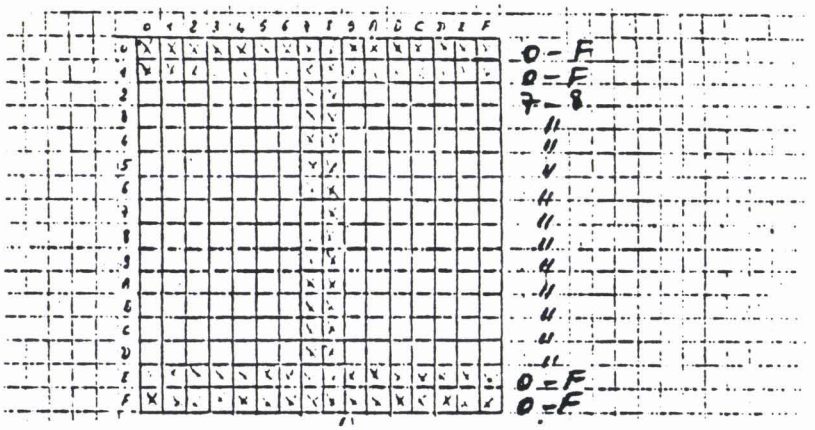
Exemple : Une image quelconque peut être représentée par une structure arborescente dont les feuilles constituent les objets de base composant cette image.



où {F} est l'ensemble des feuilles de l'arbre.

Dans le cas de l'interpréteur; {F} correspond à un ensemble de macro-commandes de base. $\{A_{ij\ldots k}\}$ ensemble d'images simples.

Pour illustrer nos propos nous allons prendre un exemple simple ; nous voulons créer une forme simple (BRIQUE) que nous allons manipuler dans l'image.



Nous disposons, pour ce faire, des macro-commandes suivantes qui seront détaillées en annexe :

COLORIER PAVE (numéro de pavé ; couleur ; liste de coordonnées ; fin).
DUPLIQUER PAVE (numéro de pavé, origine, numéro pavé fin).
MOV ZONE (numéro zone, couleur).
DESIGNER ZONE (numéro de zone).
DECALER BAS ZONE (numéro de zone, nombre de lignes).

Le programme FORTH correspondant est le suivant :

```
: BRIQUE (DESSIN DU MOTIF BRIQUE) HEX
  NZONE DESIGNER ZONE
  FIN OF 00 OF 01
  0E 2 DO 08 07 I LOOP
  OF OE  OF OF
(Chargement de la pile avec la liste de coordonnées)
COULEUR NPAVE COLORIERPAVE ;
```

A chaque appel de ce mot ; le motif BRIQUE est dessiner sur l'écran à l'endroit désigné par le numéro de pavé NPAVE.

COULEUR NPAVE, NZONE sont des variables définies auparavant ou des valeurs immédiates.

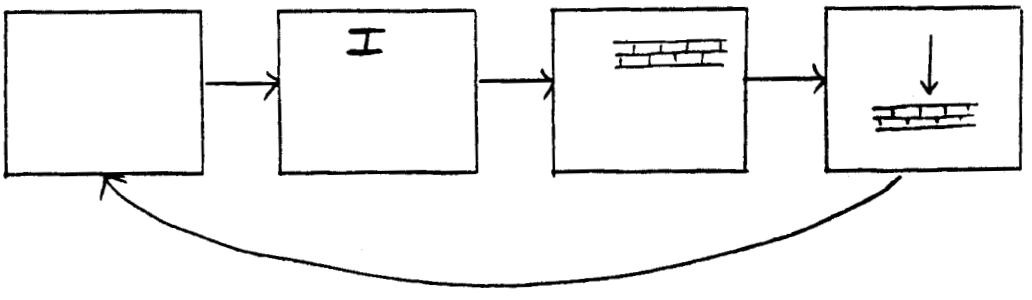
La manipulation peut se faire de la façon suivante :

```
: MANIPULATION
BEGIN COULEUR NZONE MOVZONE
(couleur du fond)

BRIQUE NPAVE2 NPAVE1 DUPLIQUER PAVE
NLIGNES NZONE DECALER BAS ZONE
TEST UNTIL ;
```

où NPAVE2, NPAVE1, COULEUR, NZONE doivent être prédéfinis ou immédiats.

TEST : condition d'arrêt.



L'exécution se fait par simple appel de mot MANIPULATION.

Notons aussi que cette opération aurait pu se faire en mode exécution ; les octets correspondant aux macro-commandes sont, dans ce cas, émis vers le processeur maître au fur et à mesure de leur entrée au clavier.

V.5. Conclusion

Tel qu'il est conçu et réalisé l'interpréteur permet de se greffer sur les macro-commandes et permettre ainsi leur exploitation. Le problème qui peut surgir d'une telle implémentation est celui de l'utilisation de la pile ainsi que la production d'opérations en notation polonaise post-fixée. Cependant, cet obstacle peut être habilement détourné en ajoutant un certain nombre de mots faisant passer de la notation infixée en notation post-fixée, ceci peut être surmonté aussi grâce à une certaine habitude. Bon nombre de programmeurs utilisent FORTH maintenant et il existe sur la majorité des micro-ordinateurs individuels ce qui prouve que ces obstacles ne sont pas insurmontables.

Pour notre application, ce langage présente des atouts certains :

- Grande rapidité qui approche celle de l'assembleur puisqu'on peut descendre à des niveaux très bas manipulant du code machine.

- Intéractivité permettant la mise au point en "pas à pas" des programmes. Ce point est très important en synthèse d'images car il permet de travailler directement sur l'image affichée pour la modifier ou la compléter.

CONCLUSION GENERALE

Le système réalisé se propose une application dans le domaine de la synthèse et de la manipulation d'images. Ceci impose que certains éléments soient spécifiques au domaine graphique c'est le cas du moniteur vidéo, ainsi que le jeu de macro-commandes.

Cependant, la configuration et les performances de ce système le rendent apte dans d'autres domaines ; c'est le cas notamment des applications suivantes :

- Reconnaissances de formes.
- Simulation de processus parallèles en temps réel .
- Intelligence artificielle.
- Etc...

Chaque application nécessite, cependant, un logiciel ainsi que des éléments matériels spécifiques. Dans le cas de reconnaissance d'images par exemple, le système devra être doté des éléments suivants :

* Capteur d'images : Caméra de télévision, appareil photographique, etc...

Ceci nécessite bien sûr un interface matériel permettant le passage d'un ensemble analogique à un ensemble digital.

* Logiciel spécifique au traitement d'images : Reconnaissance d'éléments d'images, filtrage, manipulation,...

L'architecture de la machine présente plusieurs avantages et inconvénients :

- Les différents sites sont à base d'éléments identiques. Techniquement, ce sont des cartes stéréotypées. Ceci permet d'avoir des sites banalisés d'une part et assure une maintenance facile du système (remplacement d'une carte par une autre par exemple en cas de panne).

- Un autre élément important consiste dans la faculté d'adaptation du système à divers modes de fonctionnement (SIMD, MIMD) selon le logiciel.

- La micro-programmation confère à la machine une grande puissance dans l'exécution ainsi que la spécialisation dans un domaine privilégié grâce au jeu de macro-commandes.

Le système atteint ses performances maximales pour des traitements nécessitant un faible couplage entre les différents processeurs ; la micro-programmation de certaines tâches qui réclament un grand flux de contrôle est difficile et pénalise les performances du système (exemple : rotation,...)

L'extension de la machine par ajout d'un ou plusieurs processeurs ne peut se faire qu'au prix des modifications suivantes :

a - Reconfiguration des réseaux de communication et de routage.

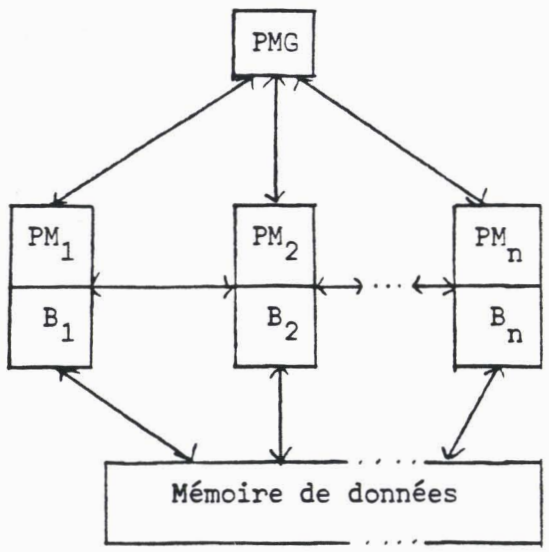
b - Reconfiguration des outils matériels de synchronisation, car tel qu'ils sont conçus actuellement ils sont spécifiques au réseau du système.

c - Redécoupage de la mémoire d'images.

d - Remise en cause du noyau de logiciel de gestion des macro-commandes.

e - La complexité du contrôle augmente avec le nombre d'éléments contrôlés, au bout d'un certain nombre de processeurs les performances de la machine diminuent.

Nous pouvons cependant imaginer une extension plus modulaire qui consisterait à considérer le réseau de processeurs comme module de base ; dans ce cas chaque module (B_i) dispose de son propre module maître (PM_i).



Un module de contrôle général PMG permettrait le contrôle de l'ensemble. Plusieurs constatations peuvent être faites au sujet d'une telle architecture.

- Grande extensibilité.
- Redécoupage de la mémoire de données.
- Nécessité de réseaux de communication et de synchronisation entre les blocs.
- Contrôle hiérarchisé à trois niveaux (PMG, PM_i, B_k).
- Une telle architecture n'est envisageable que dans le cadre d'une intégration VLSI.

Dans le cadre de l'application en cours, plusieurs voies de recherche peuvent être envisagées. Elles concernent des aspects logiciels et matériels tels que :

- Développement de logiciels de haut niveau de synthèse et d'animation d'images. Ces logiciels pouvant être développés sur le noyau de base étudié dans les chapitres IV et V. On peut envisager l'étude et l'application à la machine de systèmes de gestion de bases de données graphiques par exemple...

- Connexion à la machine d'un ensemble d'unités de communication (Light-pen, souris,...).

- Communication avec un élément de stockage d'images tel que : magnétoscope, appareil photo,...

ANNEXES :

Nous présentons ci-après quelques macro-commandes graphiques dont nous avons réalisé quelques unes actuellement. La liste peut être étendue.

Nous avons adopté le format de représentation suivant :

nom macro, x $\{P_i\}$

où x désigne un objet de base (Pavé, zone,...)

$\{P_i\}$ ensemble de paramètres (cf. Chapitre III).

Exemple : MOVI, X O_i, V

peut correspondre à : MOVPAVE P_i, V

où P_i : numéro de pavé

MOVZONE Z_i, V

Z_i : numéro de zone

V : valeur immédiate

1. AFFECTATION (ou transfert)

- MVI, X O_i, V

$V \rightarrow O_i$

- MOV, X O_1, O_2

$O_2 \rightarrow O_1$

- DUPLIQUER P_1, P_2

dupliquer le pavé P_1 depuis P_1 jusqu'à P_2.

2. ARITHMETIQUES

- ADDI, X O_i, V

$O_i \leftarrow O_i + V$

- ADD, X O_1, O_2

$O_1 \leftarrow O_1 + O_2$

- SUBI, X O_i, V

$O_i \leftarrow O_i - V$

- SUB, X O_1, O_2

$O_1 \leftarrow O_1 - O_2$

3. LOGIQUES

- ANDI, X O_i, V

$O_i \leftarrow O_i$ et V

- AND, X O_1, O_2

$O_1 \leftarrow O_1$ et O_2

- ORI, X O_i, V

$O_i \leftarrow O_i$ ou V

- OR, X O_1, O_2

$O_1 \leftarrow O_1$ ou O_2

- XORI, X O_i, V

$O_i \leftarrow O_i \oplus V$

- XOR, X O_1, O_2

$O_1 \leftarrow O_1 \oplus O_2$

4. COMBINAISON

- LDIZ, X O_i, V (chargement si zéro)

  O_i := si O_i = 0 alors V sinon O_i fsi

- LDINZ, X O_i, V (Chargement si non zéro)

  O_i := si O_i = 0 alors O_i sinon V fsi

- ADIH, X O_i, V (Addition immédiate bornée haut)

  O_i := si $(O_i + V) \leq 255$ alors $O_i + V$ sinon 255 fsi

- SUBIL, X O_i, V (Soustraction immédiate bornée bas)

  O_i := si $(O_i - V) > 0$ alors $O_i - V$ sinon 0 fsi

- SOR, X O_i, V (Supérieur ou rien)

  O_i := si $(O_i > V)$ alors O_i sinon 0 fsi

- EOR, X O_i, V (Egal ou rien)

  O_i := si $(O_i = V)$ alors O_i sinon 0 fsi

- IOR, X O_i, V (Inférieur ou rien)

  O_i := si $(O_i < V)$ alors O_i sinon 0 fsi

5. DESSINS

- COLORIER, X O_i ; $\{C_i, P_i\}$
  coloriage de O_i ligne par ligne
  P_i ::= NL, Cd, Cf
  NL : numéro de ligne
  Cd : début de coloriage (numéro de colonne)
  Cf : fin de coloriage (numéro de colonne)
  C_i : couleur

- MOTIF NM[, V]

  affichage du motif NM sur l'écran.

  [V] facultatif. Concerne les motifs paramétrables.

6. FIGURES GEOMETRIQUES

- SEGMENT, X P_1, P_2
  Tracé du segment d'extrémités P_1, P_2

- CERCLE, X O, R
  O : centre. R : rayon

- RECTANGLE, X P_1, P_2

- CARRE, X P_1, P_2

7. TRANSFORMATIONS GEOMETRIQUES

- GROS, X V
  Grossissement de valeur V

- RETR, X V
  Rétrécissement de valeur V

- DECALERBAS, X V
  Décalage horizontal vers le bas de valeur V

- DECALERHAUT, X V
  Décalage vers le haut de valeur V.

8. DESIGNATION

- DESIGNER, X P

S'il s'agit d'une partition P est un couple $\{P_1, P_2\}$ de point s opposés clôturant celle-ci. Sinon P est le numéro d'une zone ou d'un pavé...

8X300

DESCRIPTION

The Signetics 8X300 is a high-speed microprocessor implemented with bipolar Schottky technology. As the central processing unit, CPU, it allows 16-bit instructions to be fetched, decoded and executed in 250ns.

Instructions operate on 8-bit, parallel data. Logic is distributed along the data path within the microprocessor. Input data can be rotated and masked before being subject to an arithmetic or logical operation; and output data can be shifted and merged with the input data, before being output to external logic. This allows 1- to 8-bit I/O and data memory fields to be accessed and processed in a single instruction cycle.

Program storage interface

Program Storage is connected to the A0-A12 (A12 is least significant bit) and I0-I15 signal lines. An address output on A0-A12 identifies one 16-bit instruction word in program storage. The instruction word is subsequently input on I0-I15 and defines the operations which are to follow. See Fig. 1.

The Signetics 82S115 PROM, or any TTL compatible memory, may be used for program storage.

The IV bus

All data input to or output from the 8X300 goes via the Interface Vector (IV) bus. This IV bus serves both as an address and data bus and is accompanied by control signals to determine its function. Being an 8-bit bus, it has the capability to address up to 256 I/O ports (IV bytes). The input/output facilities are further expanded by selection of Left Bank (LB) or Right Bank (RB) addresses, giving a total of 512 addressable IV bytes.

When the 8X300 is required to accept data from or send data to a particular IV byte, it must first enable the IV byte. An IV byte is enabled when its address is presented on the IV bus and the bus control signals indicate that the present information is an address on the required bank. The IV byte will remain enabled until another IV byte on the same bank is enabled, at which time it becomes disabled.

The SC and WC signals are typically used to distinguish between I/O data and I/O address information as follows:

SC	WC	
1	0	I/O address is being output on IV bus
0	1	I/O data is being output on IV bus
0	0	I/O data is expected on the IV, bus, as input to the microprocessor
1	1	Not generated by the microprocessor.

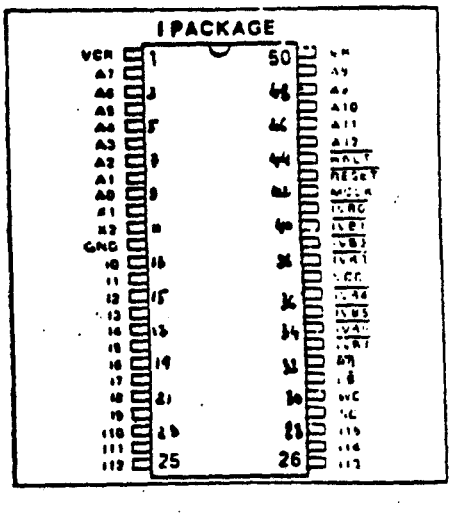
The Signetics 82SXXX series RAM, and the BT32/33/35/35 I/O ports may be attached to the IV bus.

Because the Left and Right Banks are independent, one IV byte on each Bank can be active (enabled) simultaneously. Data input from, or output to the IV bus implies data I/O to the active IV byte on the Bank specified by the instruction causing the I/O action. The most significant bit of all data is bit 0, the left-most bit of data bytes.

PIN DESCRIPTION

PIN	SYMBOL	NAME AND FUNCTION	TYPE
2-9, 45-49	A0-A12:	Instruction address lines. A high level equals "1." These outputs directly address up to 8192 words of program storage. A12 is least significant bit.	Active high
13-28	I0-I15:	Instruction lines. A high level equals "1." Receives instructions from Program Storage. I_{15} is least significant bit.	Active high
33-36, 38-41	$\overline{IVB0}$-$\overline{IVB7}$	Interface Vector (IV) Bus. A low level equals "1." Bidirectional tri-state lines to communicate with I/O devices. $\overline{IVB7}$ is least significant bit.	Three-state Active low
42	MCLK:	Master Clock. Output to clock I/O devices, and/or provide synchronization for external logic	Active high
30	WC:	Write Command. High level output indicates data is being output on the IV Bus.	Active high
29	SC:	Select Command. High level output indicates that an address is being output on the IV Bus.	Active high
31	$\overline{LB}$:	Left Bank. Low level output to enable one of two sets of I/O devices ($\overline{LB}$ is the complement of RB).	Active low
32	$\overline{RB}$:	Right Bank. Low level output to enable one of two sets of I/O devices ($\overline{RB}$ is the complement of $\overline{LB}$).	Active low
44	$\overline{HALT}$:	Low level is input to stop the Interpreter.	Active low
43	$\overline{RESET}$:	Low level is input to initialize the Interpreter.	Active low
10-11	X1, X2:	Inputs for an external frequency determining crystal. May also be interfaced to logic or test equipment.	
50	VR	Reference voltage to Pass Transistor.	
1	VCR	Regulated output voltage from Pass Transistor.	
37	VCC:	5V power connection.	
12	GND:	Ground.	

PIN CONFIGURATION



8X300 SYSTEM TIMING

The 8X300 processor cycle is built up of two periods of the clock at inputs X1 and X2. A processor cycle can be divided into four phases of half a clock period each. Figure 6 shows the timing diagram. Because most of the signal timings are specified with respect to the clock edges from which they are derived, these values do not change when the clock frequency is changed.

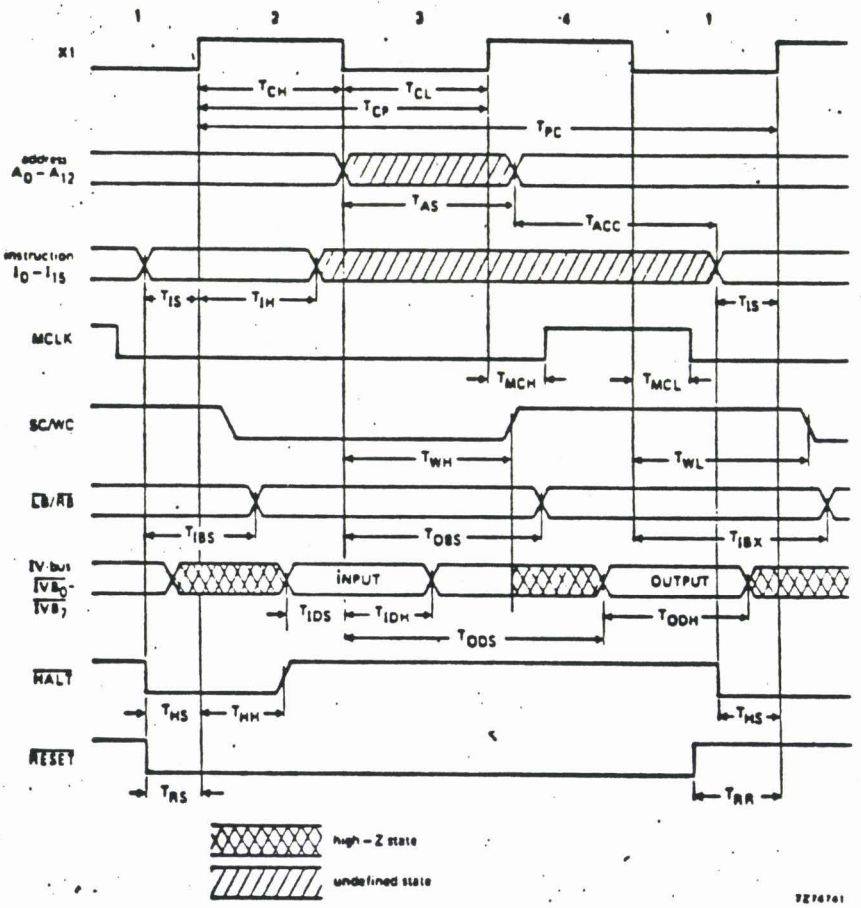


Fig. 6. System timing diagram.

INSTRUCTION OPERATION IN THE 8X300

INSTRUCTION FORMATS AND OPERAND FIELDS

An 8X300 instruction consists of a three bit operation code (OP) followed by a thirteen bit operand field. The operation code determines the class of the instruction to be performed, while the operand field provides details of the data to be processed. Figure 1 shows the general instruction format for the 8X300.

Table 1 shows the various instruction formats and the instructions that use those.

TABLE 1 Instruction formats.

Format				Instructions	
0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15		
OP	S	R	D	MOVE, ADD, AND, XOR	register to register; register to IV bus address
OP	S	L	D	MOVE, ADD, AND, XOR	register to IV bus; IV bus to register; IV bus to IV bus; IV bus to IV bus address
OP	S	I		XEC NZT	register register
OP	S	L	I	XEC NZT	IV bus IV bus
OP	D	I		XMIT	register; IV bus address
OP	D	L	I	XMIT	IV bus
OP	A			JMP	

8T32/33/35/36 Bidirectional Latched Addressable I/O Ports

TYPES

8T32 Tri-State, Synchronous User Port
8T33 Open Collector, Synchronous User Port
8T35 Open Collector, Asynchronous User Port
8T36 Tri-State, Asynchronous User Port

DESCRIPTION

The Interface Vector (IV) Byte is an 8-bit bidirectional data register designed to function as an I/O interface element in microprocessor systems. It contains 8 data latches accessible from either a microprocessor (IV) port or a user port. Separate I/O control is provided for each port. The 2 ports operate independently, except when both are attempting to input data into the IV Byte. In this case, the user port has priority.

A unique feature of the 8T32/33/35/36 IV Byte is the way in which it is addressed. Each IV Byte has an 8-bit, field programmable address, which is used to enable the microprocessor port. When the SC control signal is high, data at the microprocessor port is treated as an address. If the address matches the IV Byte's internally programmed address, the microprocessor port is enabled, allowing data transfer through it.

The port remains enabled until an address which does not match is presented, at which time the port is disabled (data transfer is inhibited). A Master Enable input (ME) can serve as a ninth address bit, allowing 512 IV Bytes to be individually selected on a bus, without decoding. The user port is accessible at all times, independent of whether or not the microprocessor port is selected.

A unique feature of this family is their ability to start up in a predetermined state. If the clock is maintained at a voltage less than .8V until the power supply reaches 3.5V, the user port will always be all logic 1 levels, while the IV port will be all logic 0 levels.

ORDERING

The 8T32/33/35/36 may be ordered in preaddressed form. To order a preaddressed IV Byte, use the following part number format:

N8TYY-XXX P

- P = F Ceramic package, NA Plastic package
- XXX = Any address from 000 through 255 (decimal) - 256 available addresses
- YY = IV Byte version (32, 33, 35, 36)

The devices may be ordered in unaddressed form by the code number: N8TYY-P. *
A stock of 8T32s and 8T36s with addresses 0 through 9 will be maintained. A small quantity of addresses 10 through 50 will also be available with a longer lead time.

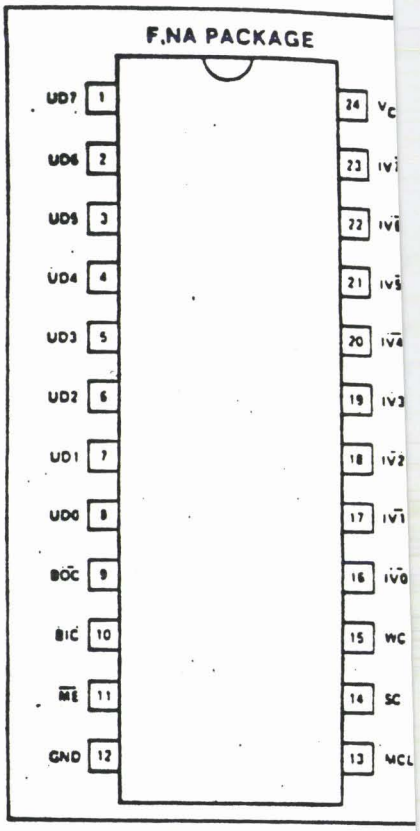
* These may be programmed by the IV byte programming kit.

FEATURES

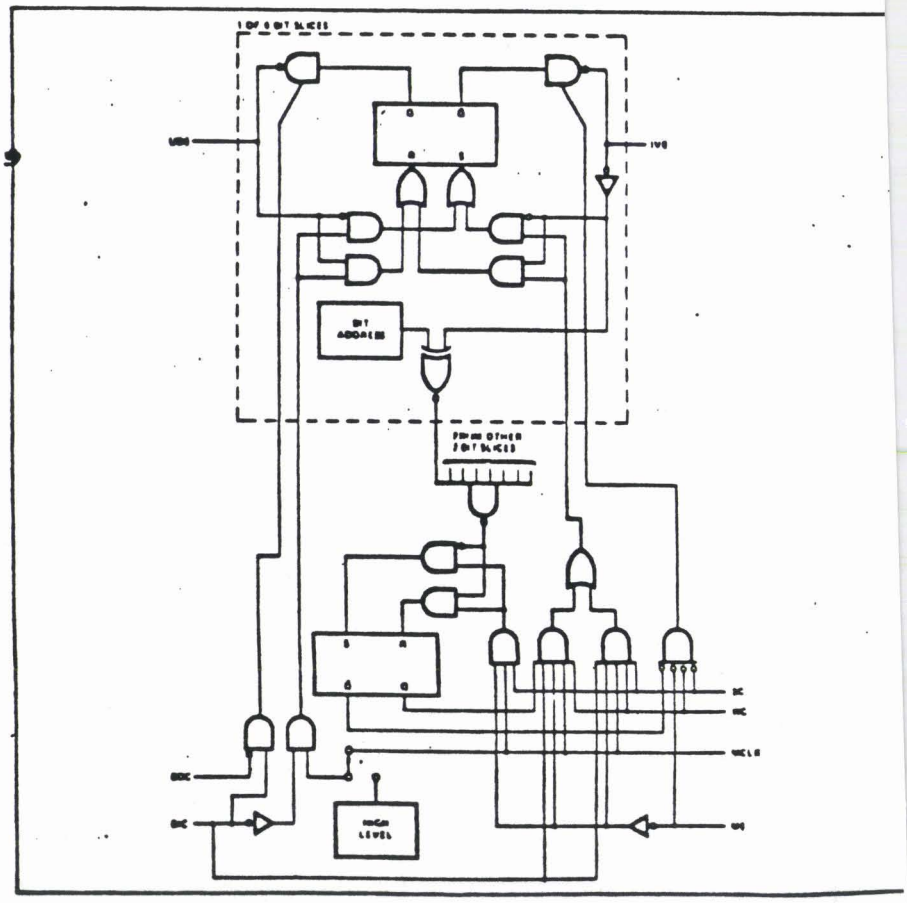
- A field-programmable address allows 1 of 512 IV Bytes on a bus to be selected, without decoders.
- Each byte has 2 ports, one to the user, the other to a microprocessor. IV Bytes are completely bidirectional.
- Ports are independent, with the user port having priority for data entry.
- A selected IV Byte de-selects itself when another IV Byte address is sensed.
- User data input available as synchronous (8T32, 8T33) or as asynchronous (8T35, 8T36) function.
- The User Data Bus is available with tri-state (8T32, 8T36) or open collector (8T33, 8T35) outputs.
- At power up, the IV Byte is not selected and the user port outputs are high.
- Tri-state TTL outputs for high drive capability.
- Directly compatible with the 8X300 interpreter.
- Operates from a single 5V power supply over a temperature range of 0° C to 70° C.

PIN CONFIGURATION

F,NA PACKAGE



BLOCK DIAGRAM



8T32/33/35/36 Bidirectional Latched Addressable I/O Ports

PIN DESCRIPTION

PIN	SYMBOL	NAME AND FUNCTION	TYPE
1-8	UD0-UD7:	User Data I/O Lines. Bidirectional data lines to communicate with user's equipment. Either tri-state or open collector outputs are available.	Active high
16-23	$\overline{IV0}$-$\overline{IV7}$:	Interface Vector (IV) Bus. Bidirectional data lines to communicate with controlling digital system (microprocessor).	Active low three-state
10	$\overline{BIC}$:	Byte Input Control. User input to control writing into the IV Byte from the User Data Lines.	Active low
9	$\overline{BOC}$:	Byte Output Control. User input to control reading from the IV Byte onto the User Data Lines.	Active low
11	$\overline{ME}$:	Master Enable. System input to enable or disable all other system inputs and outputs. It has no effect on user inputs and outputs.	Active low
15	WC:	Write Command. When WC is high and SC is low, IV Byte, if selected, stores contents of $\overline{IV0}$-$\overline{IV7}$ as data.	Active high
14	SC:	Select Command. When SC is high and WC is low, data on $\overline{IV0}$-$\overline{IV7}$ is interpreted as an address. IV Byte selects itself if its address is identical to IV bus data; it de-selects itself otherwise.	Active high
13	MCLK:	Master Clock. Input to strobe data into the latches. See function tables for details.	Active high
24	VCC:	5V power connection.	
12	GND:	Ground.	

$\overline{BIC}$	$\overline{BOC}$	MCLK	USER DATA BUS FUNCTION	
			8T32, 8T33	8T35, 8T36
H	L	X	Output Data	Output Data
L	X	H	Input Data	Input Data
L	X	L	Inactive	Input Data
H	H	X	Inactive	Inactive

H = High Level L = Low Level X = Don't care

Table 1 USER PORT CONTROL FUNCTION

$\overline{ME}$	SC	WC	MCLK	$\overline{BIC}$	STATUS LATCH	IV BUS FUNCTION
L	L	L	X	X	SET	Output Data
L	L	H	H	H	SET	Input Data
L	H	L	H	X	X	Input Address
L	H	H	H	L	X	Input Address
L	H	H	H	H	X	Input Data and Address
L	X	H	L	X	X	Inactive
L	H	X	L	X	X	Inactive
L	L	H	H	L	X	Inactive
L	L	X	X	X	Not Set	Inactive
H	X	X	X	X	X	Inactive

Table 2 MICROPROCESSOR PORT CONTROL FUNCTION

USER DATA BUS CONTROL

The activity of the User Data Bus is controlled by the BIC and BOC inputs as shown in Table 1.

For the 8T32 and 8T33, User Data Input is a synchronous function with MCLK. A low level on the BIC input allows data on the User Data Bus to be written into the Data Latches only if MCLK is at a high level. For the 8T35 and 8T36, User Data Input is an asynchronous function. A low level on the BIC input allows data on the User Data Bus to be latched regardless of the level of the MCLK input. Note that when 8T35 or 8T36 IV Bytes are used with the 8X300 Interpreter care must be taken to insure that the IV Bus is stable when it is being read by the 8X300 Interpreter.

To avoid conflicts at the Data Latches, input from the Microprocessor Port is inhibited when BIC is at a low level. Under all other conditions the 2 ports operate independently.

INTERFACE VECTOR BUS CONTROL

As is shown in Table 2, the activity of the microprocessor port (IV Bus) is controlled by the ME, SC, WC and BIC inputs, as well as the state of an internal status latch. BIC is included to show user port priority over the microprocessor port for data input.

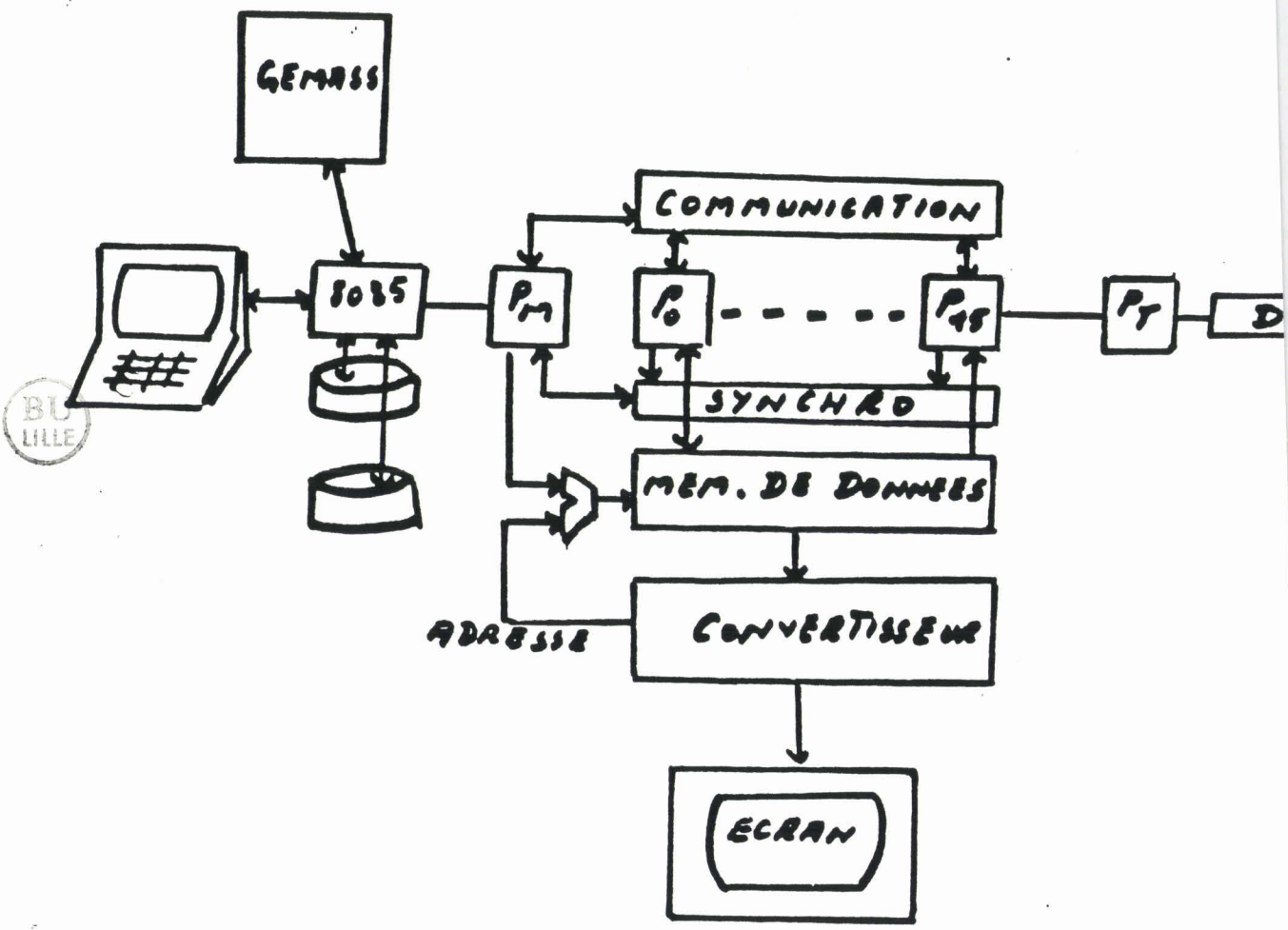
Each IV Byte's status latch stores the result of the most recent IV Byte select; it is set when the IV Byte's internal address matches the IV Bus. It is cleared when an address that differs from the internal address is presented on the IV Bus.

In normal operation, the state of the status latch acts like a master enable; the microprocessor port can transfer data only when the status latch is set.

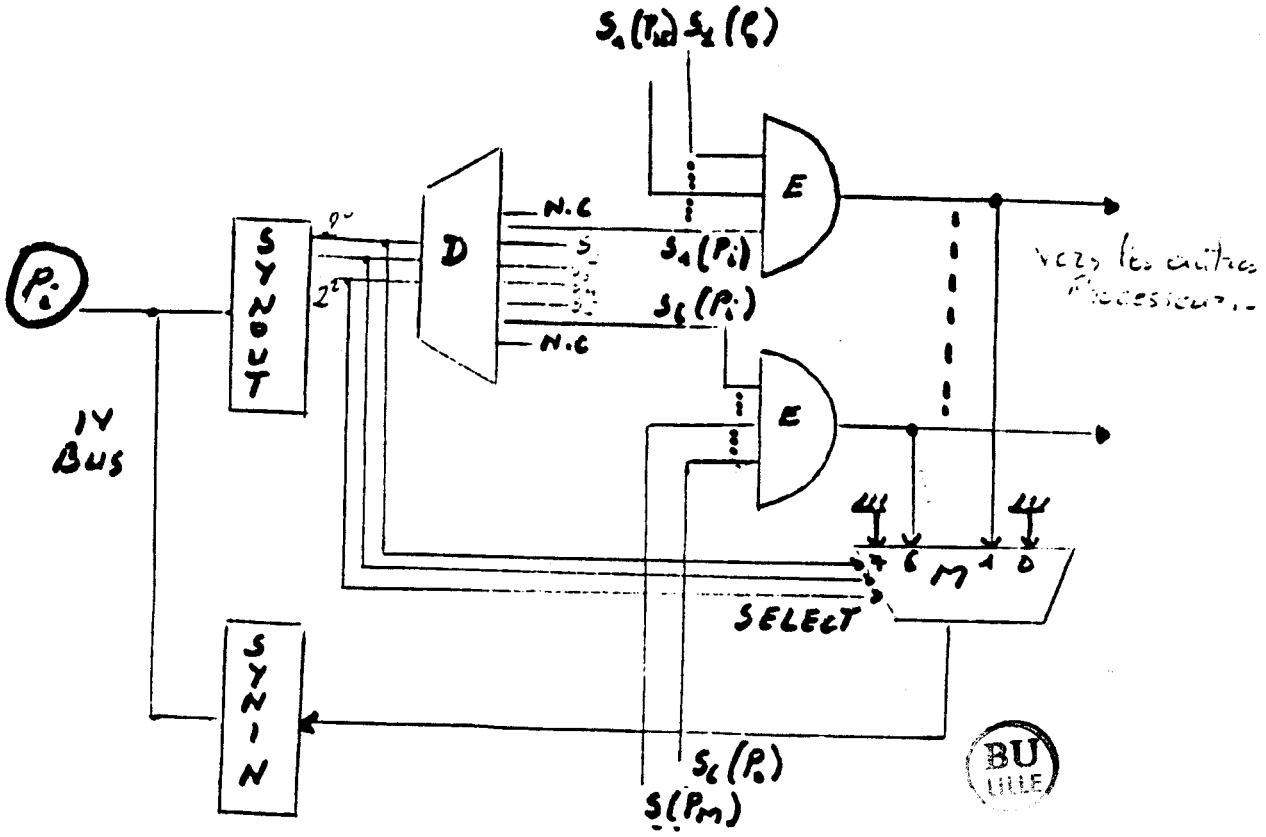
When SC and WC are both high, data on the IV Bus is accepted as data, whether or not the IV Byte was selected. The data is also interpreted as an address. The IV Byte sets its select status if its address matches the data read when SC and WC were both high. It resets its select status otherwise.

BUS OPERATION

Data written into the IV Byte from one port will appear inverted when read from the other port. Data written into the IV Byte from one port will not be inverted when read from the same port.



SCHEMA GLOBAL DU SYSTEME.



SYNCHRONISATION PAR GROUPES

E: Porte et logique.

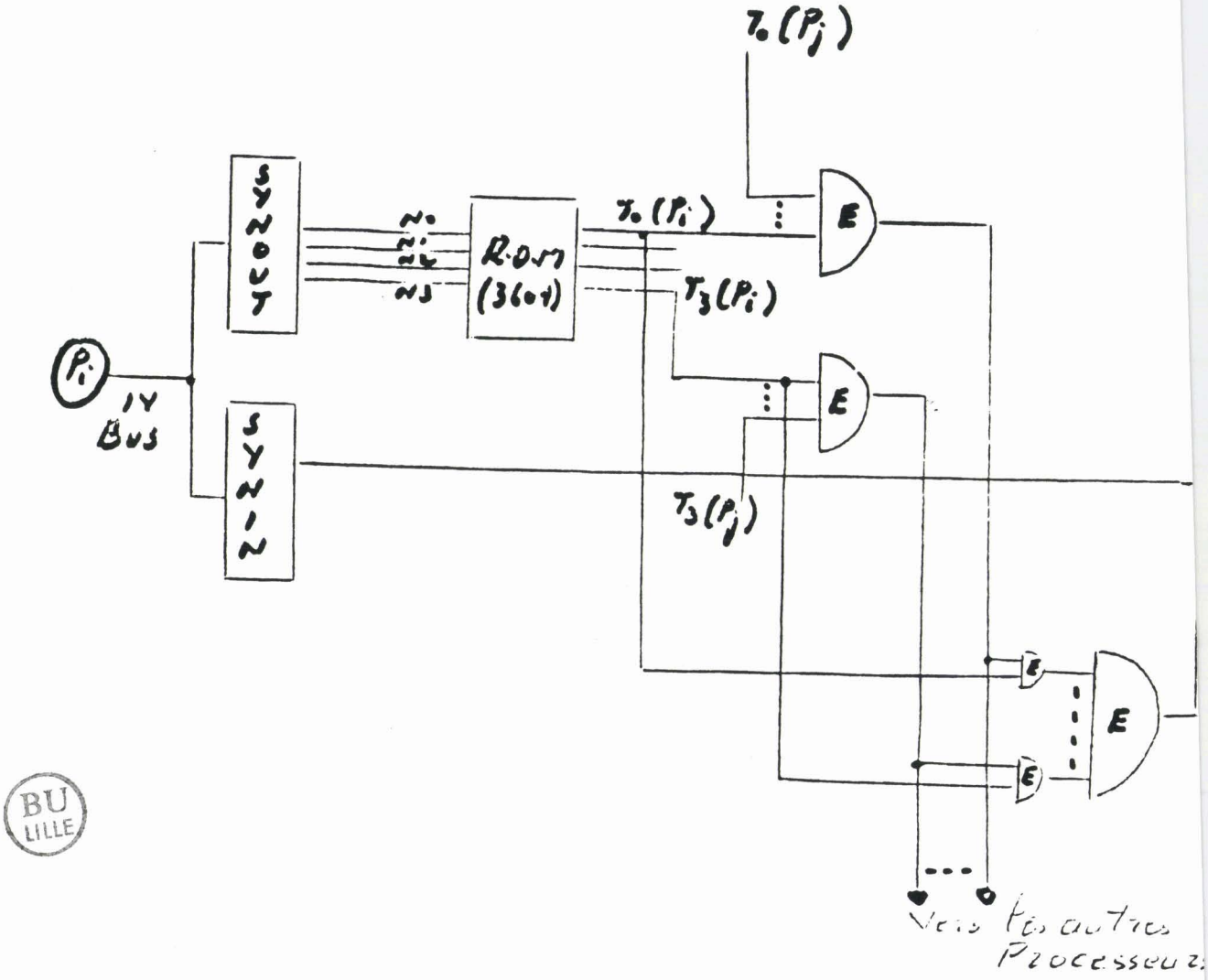
D: Decodeur (3 vers 8 : 74138).

M: multiplexeur (8 vers 1 : 74251).

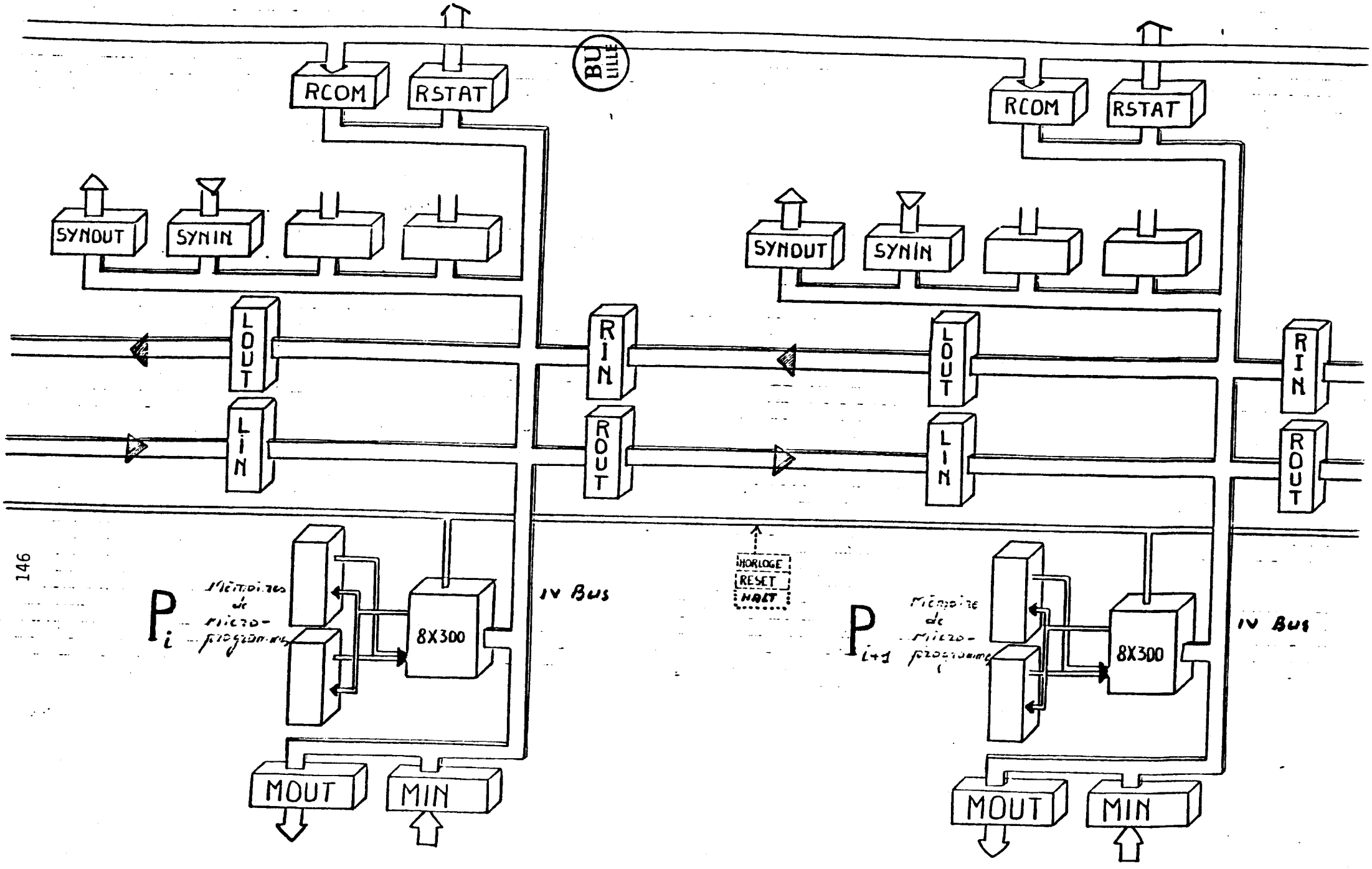
SYNOUT: Registre de synchronisation en sortie.

SYNIN: Registre de synchronisation en entrée.

SELECT: Selection de l'entrée de M.



SYNCHRONISATION PAR NOM.


RCOM
RSTAT
SYNOUT
SYNIN
LOUT
LIN
RIN
ROUT
HORLOGE
RESET
HALT
IV Bus
8X300
MOUT
MIN

BIBLIOGRAPHIE

ALL 83 - G. ALLAIN
"Images de synthèse dans les simulateurs de vol".
Bulletin de Liaison de la Recherche en Informatique et en Automatique.
INRIA n° 88, 1983.

BLI 77 - J.F. BLINN
"Models of light reflection for computer synthetized pictures".
Computer Graphics SIGGRAPH-ACM, Vol. 11, n° 2, 1977.

BOR 83 - I. BORNE
"Les objets dans LOGO".
"Journées d'Etudes sur les langages orientés objets".
BIGRE LE CAP d'AGDE, 20 Octobre 1983.

BOU 80 - P. BOULLE
"Etude et réalisation d'algorithmes pour la visualisation de scènes composées de facettes planes.
Thèse de Docteur-Ingénieur, Grenoble, 1980.

BRA 81 - A. BRADIER
"Eléments pour la représentation des données graphiques dans un contexte intéractif".
Mémoire de D.E.A., Lille 1981.

BRO 81 - L. BRODIE
"STARTING FORTH".
Prentice-Hall, 1981.

CAR 82 - C. CARREZ
"Outil de partage de fichiers dans un Laboratoire de Recherche en architecture".
Congrès AFCET, Lille 1982.

C.E.I. 82- CEA-EDF-INRIA
"La réalisation de logiciels graphiques intéractifs".
EYROLLES 1982.

COI 82- P. COINTE
"Une implémentation de SMALLTALK en VLISP".
Publication EDF 1982.

COI 83- P. COINTE
"Comprendre SMALLTALK, penser SMALLTALK".
Journée BIGRE CAP d'AGDE, 20 Octobre 1983.

COM 81- V. CORDONNIER, L. MOUSSU
"The MAP project : An associative processor for speech processing".
10th International Conference on Parallel Processing. Bellaire, Michigan
August 25-28, 1981.

COR 78- V. CORDONNIER
"Architecture pipeline".
Support du cours IRIA sur le traitement parallèle, Lille 29 Mai-2 Juin 1978.

COR 83- V. CORDONNIER
"Noyau pour l'écriture de micro-programmes parallèles".
Publication interne E.R.A. Février 1983.

COR 83- V. CORDONNIER
"Un modèle associatif de synchronisation dans un système multiprocesseur".
IASTED Symposium, Lille, Mars 1983.

CRO 76- CROCUS
"Système d'exploitation des ordinateurs".
DUNOD 1976.

CUL 83- CULOTTI
"Langage d'animation d'images".
Mémoire de D.E.A., Lille 1983.

DEL 79 - E. DELATTRE
"Contribution à la répartition d'un système à structure de domaines sur un réseau local de micro-ordinateurs faiblement couplés".
Thèse de Docteur - Ingénieur, Lille 1979.

EEKP 79 - R. ECKERT, G. ENDERLE, K. KANSKY, F.J. PRESTER
"GKS 79 : Proposal of a standard for a GKS".
Congrès Eurographics, Bolosna, Septembre 1979.

EEK 80 - J. ENCARNACAU, G. ENDERLE, K. KANSAY
"The workstation concept of GKS and the resulting conceptual differences to the GSPC core system".
Computer Graphics SIGGRAPH-ACM, n° 3, Août 1980.

FAK 83 - E.T. FATHI, M. KRIEGER
"Multiple microprocessor systems : wath, why, who".
Computer March 1983.

FER 81 - N. FERREIRA
"Conception et réalisation d'un système intéractif pour la synthèse d'images réalistes : HELIOS".
Thèse de Docteur - Ingénieur, Grenoble, Septembre 1981.

FER 82 - L. FERROUDJI
"Etude de tables de couleurs commutables pour un système de synthèse d'images".
Mémoire d'Ingénieur INI. ALGER, 1982.

FLY 72 - M.J. FLYNN
"Some Computer Organization and their Effectiveness".
I.E.E.E. Transactions on Computer, Septembre 1972, pp. 59-76.

FOV 82 - J.D. FOLEY, A. VANDAM
"Fundamentals of interactive Computer Graphics".
Addison-Welsey, 1982.

FRA 81 M.A. FRANKLIN
"VLSI Performance comparison of Banyan and Crosshan communication networks".
I.E.E.E. Transactions on Computer, Vol. C-30, n° 4, April 1981.

GOC 82 - M.F. GORDON, S.V. COPE
"Coprocessing to ease the Graphics burden".
Computer design, July 1982, pp. 147-152.

GSP 79
Status report of the GSPC.
Computer Graphics, SIGGRAPH-ACM, Vol. 13, n° 3, 1979.

GOM 81 - G. GONCALVES, Y. MERIADEC
"Processeur trace sur MAP".
Mémoire d'Ingénieur ISEN, Lille 1981.

GRA 80 - M. GRAVE
"Etude d'un noyau de système d'images. Application à la visualisation de scènes tridimensionnelles".
Thèse de Docteur Ingénieur, Lille, Décembre 1980.

GRA 83 - M. GRAVE
Cours d'informatique graphique
INRIA 1983.

HER 79 - B. HERZOG
"Functionnal specification for a core graphics system".
Ecole d'Eté d'Informatique, 1979.

HIG 73 - L.C. HIGHIE
"Super Computer Architecture".
Computer, Vol. 6, n° 12, Décembre 1973.

HLSM 82 - L.S. HAYNESS, R.L. LAU, D.P. SIEWIOREK, D.W. MIZELL
"A survey of highly parallel computing".
Computer, January 1982.

KIL 81 - A.C. KILGOUR
"A hierarchical model of a graphic system".
Computet Graphics, Avril 1981.

LAM - L. LAMPORT
"The synchronisation of independant processes".
Acta Informatica, Vol. 7.

LLM 78 - A. LEDUC, M. LUCAS, F. MARTINEZ
"Conception et réalisation d'un logiciel graphique intéractif indépendan du contexte d'utilisation. Le logiciel de base GRIGRI".
RAIRO Informatique, Vol. 12, n° 2, 1978.

LOE 81 - R.G. LOELIGER
"Threaded Interpretive langages".
Byte Books, 1981.

LUC 77 - M. LUCAS
"Contribution à l'étude des techniques de communication graphiques avec un ordinateur. Eléments de base des logiciels graphiques intéractifs
Thèse de Docteur-ès-Sciences, Grenoble 1977.

MAR 82 - F. MARTINEZ
"Vers une approche systématique de la synthèse d'images. Aspects logiciel et matériel".
Thèse de Docteur-ès-Sciences, Grenoble 1982.

MAR 83 - F. MARTINEZ
"Le projet HELIOS".
Bulletin de la Recherche en Informatique et en Automatique INRIA 88/1983.

MER 79 - M. MERIAUX
Etude et réalisation d'un terminal graphique couleur tridimensionnel fonctionnant par tâches".
Thèse de Docteur-Ingénieur, Lille 1979.

MER 82 - M. MERIAUX
"La programmation des couleurs en Informatique graphique".
Publication interne ERA n° 24, Décembre 1982.

MIC 80 - MICADO
"Matériels et logiciels pour la C.A.O.".
1980.

MOL 76 - P. MORVAN, M. LUCAS
"Images et ordinateur, introduction à l'infographie intéractive".
LAROUSSE 1976.

NES 79 - W. NEWMAN, R.F. SPROULL
"Principles of interactive Computer Graphics".
2nd edition Mc. Graw-Hill, New-York, 1979.

NNS 72 - M.E. NEWELL, R.G. NEWELL, T.L. SANCHA
"A new approach to the shaded picture problem".
Proc. of ACM national Conférence, Vol. 1, Boston Août 1972.

OHL 80 - M.R. OHLSON
"Dësign langage for interactive graphics applications systems".
PHD thesis, Texas ASM University, August 1980.

OLE 82 - R. OLEJNICK
"Réalisme dans les images générées par ordinateur".
Mémoire de D.E.A., Lille 1982.

PAP 71 - S. PAPERT
"A computer laboratory for elementary schools LOGO memo n° 1".
A.I. Laboratory, M.I.T., Octobre 1971.

PRE 79 - D. PRENER
"Large multimicroprocessor system".
Microprocessor and microsystem, Vol. 13, n° 6, July-August 1979.

RED 81 - M. REDJIMI
"Architecture de MAP".
Mémoire d'Ingénieur CERI 1981.

RED 84 - M. REDJIMI
"Conception d'un système parallèle pour le graphique".
BIARRITZ 21-25 Mai 1984.

SCH 80 - R.A. SCHUMACKER
"A new visual system architecture".
Proc. 2th IITEC, Novembre 1980.

SIN 80 - M. SINT
"Design of an ALGOL 68 extension for graphics".
Cours Européen d'Informatique Graphique, Nantes 1980.

SST 83 - W.P. SALMAN, O. TISSERAND, B. TOULOUT
"FORTH".
EYROLLES 1983.

TSE 81 - TSE YUNG FENG
"A survey of interconnection networks".
Computer, Décembre 1981, Vol. 14, n° 12.

VAN - D. VANDORPE
"Introduction à la C.A.O.".
Université de Lyon.

WHI 80 - T. WHITTED
"An improved illumination model for shaded display".
CACM, Vol. 23, n° 6, Juin 1980.

Documentations techniques :

- INTEL, ZENDEX : Systèmes multibus.

- SIGNETICS : 8 × 300 et circuits associés.

- LIF 80 : LIFEBOAT ASSOCIATES
  FORTH user manual and tutorial
  Release 2. Timin Engineering Co. 1980

- Documents d'Exploitation de M.A.P.
  Laboratoire d'Informatique U.S.T.L.1.