وزارة الجامعات
MINISTERE AUX UNIVERSITES

المدرسة الوطنية المتعددة التقنيات
ECOLE NATIONALE POLYTECHNIQUE D'ALGER

DEPARTEMENT D'ELECTRONIQUE

SUJET

CONCEPTION ET REALISATION D'UN EMULATEUR A BASE DU MICROCONTROLEUR i80186 DE INTEL

Propose par la Direction Recherche et Developpement

D.R.D / E.N.S.I

Etudié par : Mourad OUKEMOUM

Promoteurs :

Mr H.BOUSBIA

Mr ATAGHLIT

Annee 1990/91

وزارة الجامعات
MINISTERE AUX UNIVERSITES

المدرسة الوطنية المتعددة التقنيات
ECOLE NATIONALE POLYTECHNIQUE D'ALGER

DEPARTEMENT D'ELECTRONIQUE

SUJET

CONCEPTION ET REALISATION D'UN EMULATEUR A BASE DU MICROCONTROLEUR i80186 DE INTEL

Propose par la Direction Recherche et Developpement

D.R.D / E.N.S.I

Etudié par : Mourad OUKEMOUM

Promoteurs :

Mr H.BOUSBIA

Mr A.TAGHLIT

Annee 1990/91

بسم الله الرحمان الرحيم

"سنريهم ءايتنا في الآفاق و في أنفسهم حتى يتبين لهم
أنه الحق"

"Bientot Nous leur ferons voir Nos signes a tous les horizons, tout comme dans leurs propres personnes, jusqu'a ce qu'il leur devienne evident que, oui, c'est cela la verite."

Coran

REMERCIEMENTS

Je tiens à remercier vivement mon promoteur A.TAGHLIT pour sa constante disponibilité à suivre mon travail. Je remercie, également, Mr. N. BOUZA directeur de projet à l'E.N.S.I. qui m'a guidé et conseillé tout au long de ce stage, ainsi que Mr.BOUSBIA,enseignant à l'Ecole Nle Polytechnique.

Aussi, je remercie Mr. D.BERKANI, chef de département d'électronique, qui m'a fait l'honneur de présider le jury. Sans oublier Mr. A.FARAH, responsable de la post-graduation, pour ces aides et conseils.

Mes remerciements vont aussi, à toute la sympathique équipe du laboratoire, ingénieurs et stagiaires. Plus particulièrement, aux stagiaires Med. AREZKI et R. BENCHEMAM,concepteur d'un BIOS (pour carte à base du ¿80186).

Je remercie le corps travailleur du centre de calcul et de la bibliothèque.

Qu'ils soient içi remerciés tous ceux qui ont participé à ma formation.

DEDICACES

En témoignages de reconnaissances envers mes parents:

"ô mon Seigneur! Sois Misérocordieux envers eux,tout comme ils l'ont été envers moi en m'élevant lorsque j'étais petit".

Coran

A mes fréres qui combattent sur le sentier d'ALLAH.

A mes fréres qui souffrent dans les prisons pour que la parole de DIEU soit la plus haute et celle des négateurs la plus basse.

A toute ma famille.

je dedie ce modeste travail.

PRESENTATION

Dans le cadre de ses activités, le département péri-informatique de la D.R.D./E.N.S.I. a projeté de réaliser un émulateur à base du microcontrôleur 80186 de INTEL. Le projet qui m'a été confié consiste en la conception et réalisation d'une carte émulateur munie d'un minimum de fonctions hardware et d'un software assurant sa gestion.

On donnera au CHAPITRE I des généralités sur les systèmes de développement existants.

Au CHAPITRE II on définira l'émulation et les différents tests l'accompagnant.

Au CHAPITRE III sera présentée la carte EMUL186 réalisée.

Le CHAPITRE IV est consacré à la routine d'initialisation.

Enfin, au CHAPITRE V sera illustré le moniteur de gestion MON186.

SOMMAIRE

PAGE

ملخص

تم إنجاز هذه الإطروحة في مخبر مديرية البحث والتطوير(D.R.D./E.N.S.I.)
. وهذا العمل هو محاولة إنجاز ممثل قاعدته المعالج i80186 لشركة
INTEL.

والمشروع مقسم إلى قسمين : عتاد و برمجة . البطاقة المطورة تحتوي
على ذاكرات و واسطة...إلخ.
أما قسم البرمجة فهو يحتوي على مجموعة من الوظائف الخاصة بالممثل.
نستطيع أن نستعمل المعالج i80186 في عدة تطبيقات مختلفة مما يجعل
هاذا النوع من الممثل بالغ الأهمية.

ABSTRACT:

This paper has been achieved at the laboratory of Research and Developpement Direction of the E.N.S.I. It's consist's to design and realize an In-Circuit-Emulator based i80186 INTEL microcontroller.

The project is divided into two hardly bound parts : Hardware and Software.The hardware part, named "EMUL186", is composed of memories, input/output interface which permit communication between terminal or microcomputer set(in downloading mode). The software developed "MON186" is composed of some tasks that the in-circuit-emulator expected to do.

The i80186 microcontroller can be used in various applications (communication,robotics,...etc) that make this kind of emulator very interesting.

RESUME :

Le présent document décrit le travail accompli au laboratoire de la Direction Recherche et Developpement de l'Entreprise Nle des Systèmes Informatiques (D.R.D/E.N.S.I.). En effet, le projet qui m'a été confié consiste en la conception et réalisation d'un émulateur à base du microcontrôleur de INTEL i80186.

Le projet est subdivisé en deux parties, intimement liées: hardware et software. Le hardware réalisé, baptisé "EMUL186", comprends des mémoires, un interface d'entrées/sorties, assurant la communication entre la carte et un terminal ou un micro-ordinateur (en mode de téléchargement). Le sofware développé, "MON186", est un moniteur de gestion formé d'un ensemble de taches qu'un émulateur est sensé effectuer.

La réalisation d'un tel émulateur s'avère trés interessante vûe la diversité d'applications qu'offre ce chip (surtout en téléphonie, robotique,...etc).

INTRODUCTION

L'apparition du microcontrôleur i80186 de INTEL a offert aux concepteurs de produits électroniques un mono-chip trés performant. Le développement de produits à base de ce microcontrôleur nécèssite l'utilisation d'un système de développement spécifique non encore disponible au laboratoire. Vûe la diversité d'application qu'offre ce microcontrôleur, et dans le but d'acquérir un savoir faire en matière d'émulation, il a été décidé de lancer le projet d'un émulateur à base de ce mono-chip. En effet, il m'a été, alors, confié de le concevoir et de le réaliser. En l'absence de système de developpement pour les applications autour du i80186 , on a été contraint de developper notre prototype en utilisant des moyens classiques. Rendant ainsi notre tâche difficile, fastidieuse et délicate. Le seul recours valable est l'utilisation d'un analyseur logique à huit voies dispo- nible au laboratoire de recherche de l'ENSI. Ce travail constitue donc une première tentative en matière d'émulation, notons aussi, l'absence totale de documents y concernant.

CHAPITRE PREMIER :

GENERALITES SUR LES SYSTEMES DE DEVELOPPEMENT.

La croissance considérable du niveau d'intégration des composants électroniques a rendu l'utilisation des oscilloscopes et des multimètres, utilisés seuls, sans grande utilité. Cela a conduit les concepteurs à mettre au point un outil de développement répondant aux besoins de l'ingénieur concepteur de produits à base de microprocesseurs, lui facilitant, ainsi, le développement et lui procurant un gain de temps. Les outils de développement ont évolué de pair avec les nouvelles données technologiques. L'évolution de la technologie a rendu ces systèmes plus que necèssaire pour le développement de prototypes à base de microprocesseurs.

1. DEFINITION D'UN SYSTEME DE DEVELOPPEMENT :

Il se compose d'un ensemble d'éléments qui facilitent le développement aussi bien HARDWARE que SOFTWARE du prototype à réaliser et ce durant toute la période de développement. En effet la conception d'un produit à base de microprocesseur est divisée en deux parties:

- HARDWARE;
- SOFTWARE.

Une fois ces deux parties réalisées,il faut :

- tester chacune des deux parties séparement (tests unitaires, tests globaux);
- modifier éventuellement les programmes sources , suites à des erreurs logicielles;
- regénérer le programme exécutable;
- mettre le logiciel dans le materiel électronique;
- tester l'ensemble du produit contenant le logiciel et le materiel (tests dynamiques);
- programmer les EPROMS, une fois le logiciel et le materiel testés.Voir figure I.1.

La figure I.1 montre les différentes étapes de mise au point d'un prototype.

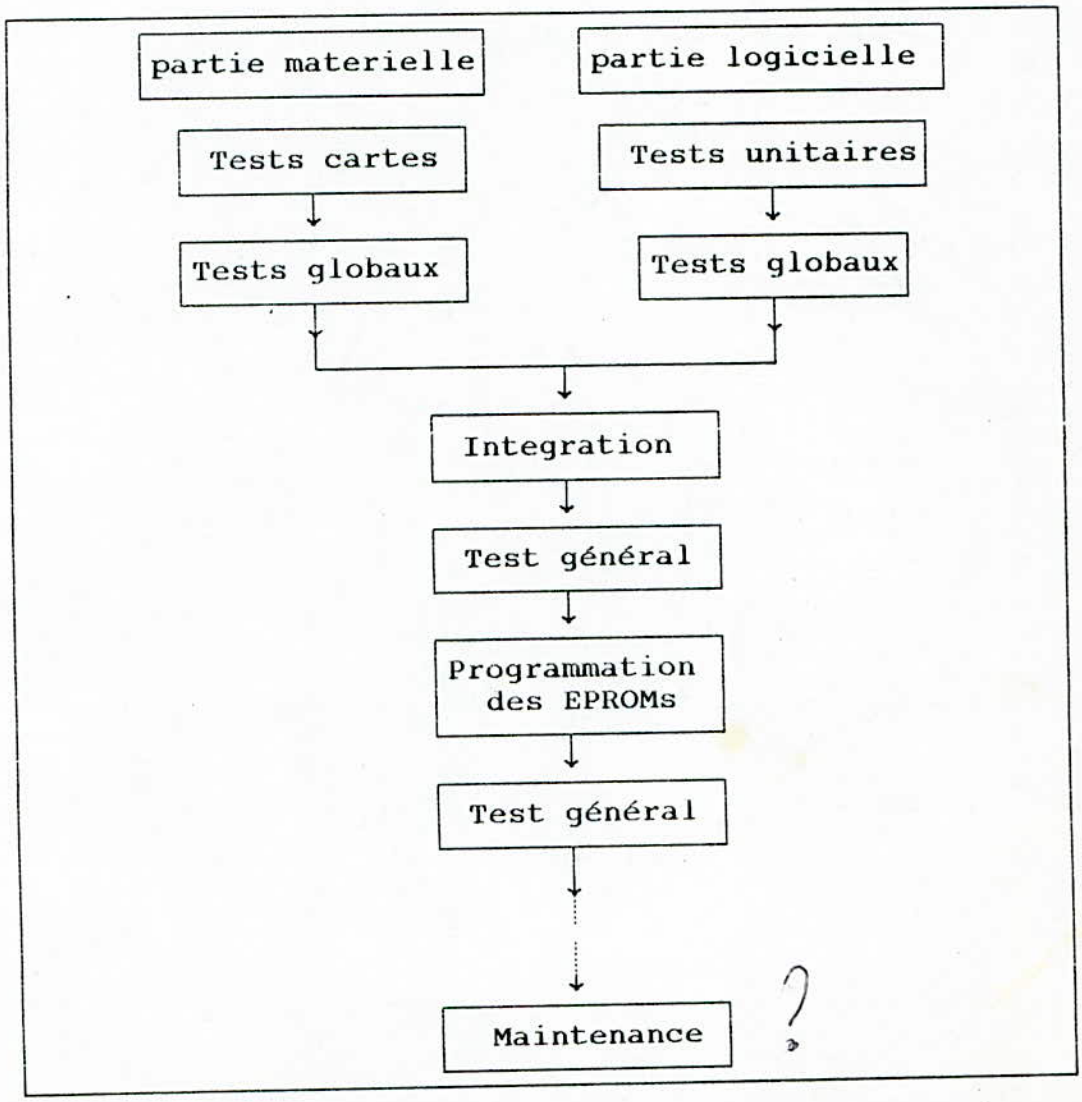


FIGURE I.1 : Schéma synoptique des étapes de mise au point d'un produit à base de microprocesseur.

En effet le système de développement intervient tout au long de ces étapes, il sert à :

a) réaliser la partie logicielle;
-création et modification de programmes sources,
-compilation ou assemblage,
-édition de liens entres différents modules.

b) tester la partie logicielle;

Exécution des tests unitaires et des tests globaux grâce à l'utilitaire DEBUGGER.

c) tester la partie materielle;

Création d'un ensemble de programmes de tests de boitiers et des cartes électroniques grâce aux utilitaires cités en a) et à la sonde d'émulation.

d) integrer le logiciel dans le materiel;

Intégration du logiciel dans le materiel et test de l'ensemble grâce à l'émulateur (utilitaire et sonde d'émulation).

e) faire le test général;

Test de l'ensemble du produit avec le programme éxecutable chargé dans la mémoire du prototype grâce à l'émulateur.

f) programmer les EPROMs ou PROMs;

g) faire la maintenance du produit par le "service aprés vente" soit par le biais de l'émulateur soit par un programme de maintenance spécifique.

2.STRUCTURE D'UN SYSTEME DE DEVELOPPEMENT :

L'environnement de réalisation de produits et de sous-produits électroniques a considérablement évolué, plus particulièrement avec l'avènement du microprocesseur. En effet, on est passé des

oscilloscopes et des multimètres,aux analyseurs logiques, aux sta tions et aux petits systèmes de développement. Voir figure I.2

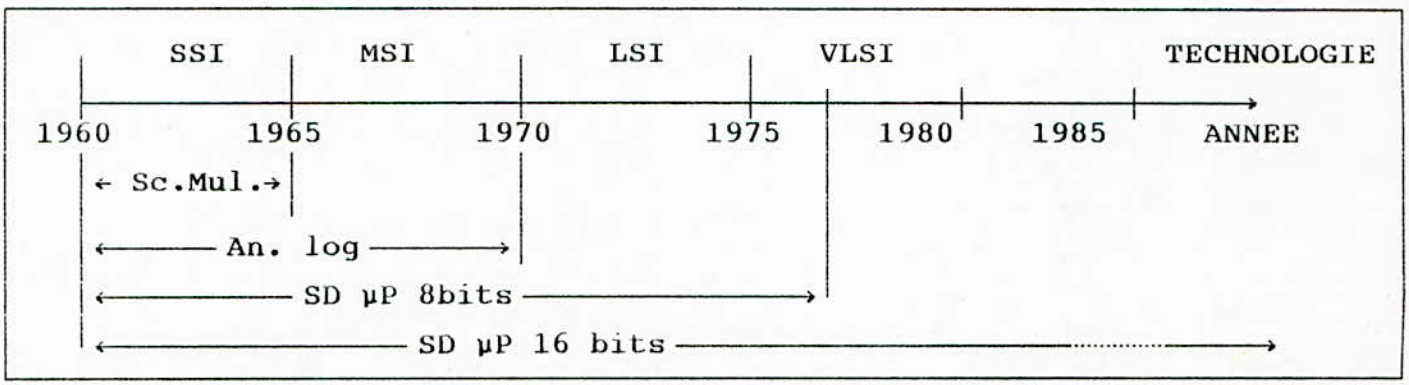


FIGURE I.2 : Croissance de la densité d'intégration et conséquences sur les outils de développement de produits électroniques.

Légendes: Sc,Mul : Scopes et multimètres;
An.Log. : Analyseur Logique;
SD : Système de développement;
µP : microprocesseur;
SSI, MSI, LSI, VLSI : Small, Médium, Large, Very Large Scale Integration.

La figure I.2 illustre la croissance de la densité d'integration et ses conséquences sur les outils de développement.

Mais peut-on prétendre se débarasser de l'osciloscope ou de l'analyseur logique? la réponse est non. A moins que la station de travail ne contienne de modules assurant la visualisation et l'analyse logique des signaux. Actuellement toute une gamme de systèmes de développement existe sur le marché. Cela va du petit kit (simple émulateur), aux stations de travail complètes (integrant des émulateurs, analyseur logique et autres...etc).

Un système de développement (SD) se présente comme un ordinateur disposant :

- d'un ensemble de périphériques (écran,clavier,imprimante,lecteur de disquettes,disque dûr...);

- d'un ensemble de cartes électroniques (ressources du système de développement);

- d'un ou plusieurs émulateurs;

- d'un analyseur logique;

- d'un programmateur d'EPROMs.

Voir figure I.3 :

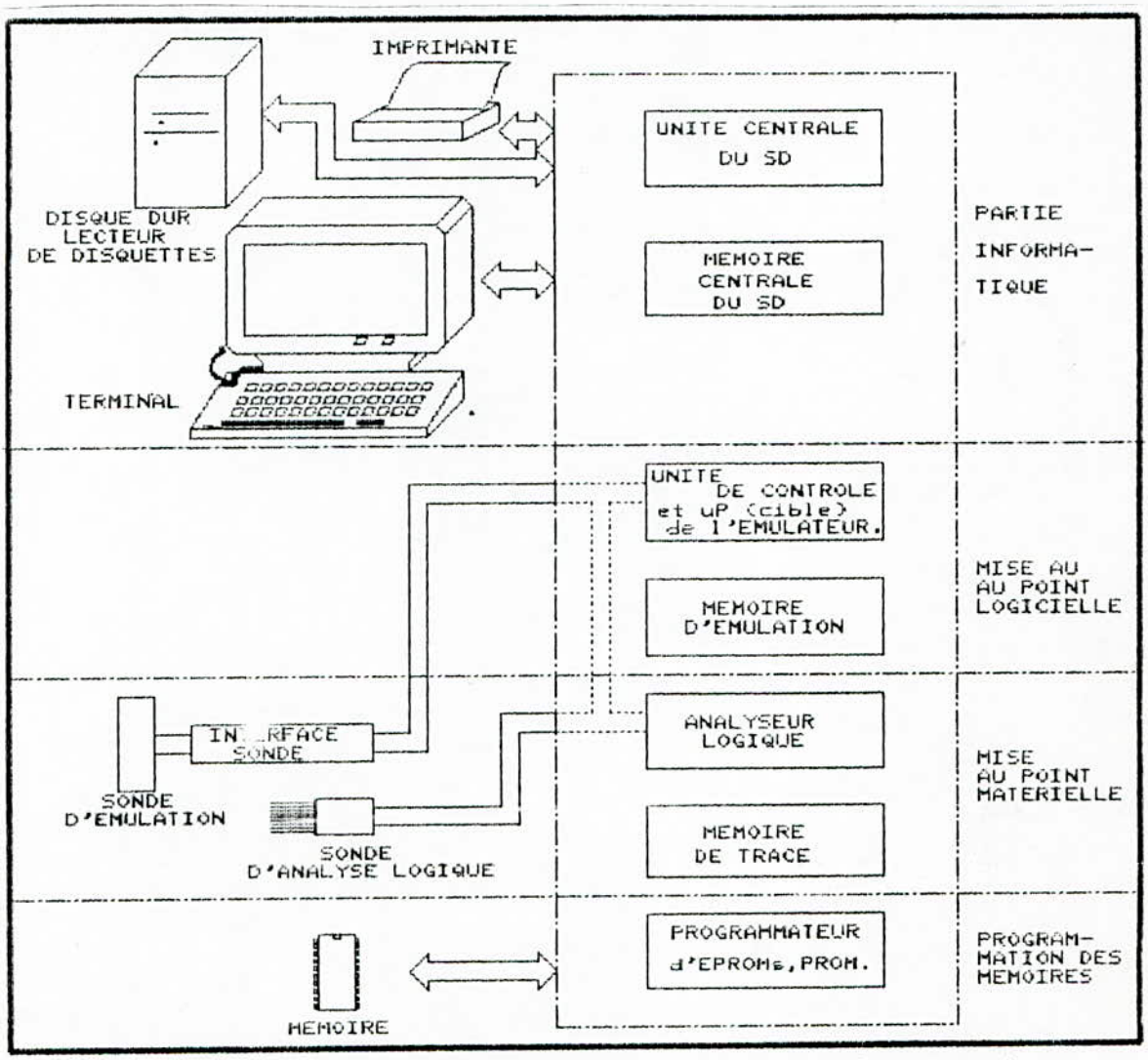


Figure I.3: Schéma d'un système de developpement.

→ Software d'un système de développement :

Le système de développement est doté d'un système d'exploitation, d'un éditeur de texte, d'un assembleur, de compilateurs (PASCAL,...etc) et d'éditeurs de liens. En outre, des logiciels de mise au point (assurant le chargement, le lancement d'un programme,son éxécution par bloc d'instructions ou par instruction.

En effet on peut éxaminer l'état des registres,d'une zone mémoire, et des ports d'entrées/sorties que l'on compare aux valeurs prévûes.Ainsi,donc,on peut détecter un éventuel "bug").

On distingue deux types d'outils de mise au point :

- Outils logiciels;

Les outils uniquement logiciels sont des programmes chargés en mémoire en même temps que le programme à tester. Ils ont pour rôle de permettre :

* l'éxamination des mémoires et des registres ;
* la modification de la mémoire et des registres ;
* le placement de points d'arrêt.

- Outils mixtes:

Avec les outils uniquement logiciels les conditions réelles d'éxecution du programme d'application ne sont pas respectées à cause des pérturbations dûes aux logiciels de mise au point que l'on exécute aussi. Pour pallier à ce défaut, on utilise, deux instruments, parfois réunis en un seul : l'émulateur et l'analyseur logique . Ces derniers ont pour fonctions "d'espionner" le comportement du microprocesseur de l'application (tout en le perturbant le moins possible), de le commander ainsi que son environnement.

TYPES DE SYSTEMES DE DEVELOPPEMENT:

Il existe quatre types de systèmes de développement :

* Systèmes de développement spécifiques à des microprocesseurs standard:

Ce sont des SD dédiés à une seule famille de microprocesseurs. Exemples: exorciser de Motorola du MC6800 (8 bits); exormac de Motorola du MC68000 (16 bits); les series INTELLEC de Intel pour le famille 8080/ 8085/ 8086/ 8088/ 80186/ 80188/ 80286...etc

* Systèmes spécifiques à des microprocesseurs personnalisés

Ce sont des SD appartenant à des constructeurs de microprocesseurs ou microcontrôleurs personnalisés.

* Systèmes de développement semi-universels:

Ces systèmes supportent deux ou plusieurs microprocesseurs de constructeurs différents les mêmes microprocesseurs sont assez comparables dans leurs architectures internes et plus particulièrement dans leur jeu d'instructions.

Exemple: SD pour les µP 8080,8085 et Z80 car on retrouve une correspondance ascendante dans le jeu d'instructions de ces trois µP issûs de deux constructeurs différents Zilog et Intel.

* Systèmes de développement universels :

Ces SD sont censés pouvoir supporter les microprocesseurs les plus utilisés .

Exemple : - Tektronics 8001/8002,8560;
- HP 64100A;
- Philips PM442 et PM4422...etc

⟶ Hardware d'un système de développement:

L'émulateur est un dispositif qui remplace le microprocesseur de l'application en cours de développement (microprocesseur cible) par un système ayant un comportement identique. Ainsi, on a accès aux fonctions internes du prototype, lors de sa mise au point tout en respectant les conditions de fonctionnement normal. De plus, on peut simuler une partie de la mémoire ou des ports d'entrée/sortie du prototype dans la mémoire d'émulation. En d'autres termes, on projette le microprocesseur cible et à travers lui, le prototype dans le système de développement par le biais de la sonde d'émulation.

Le schéma de principe d'un émulateur est donné à la figure I.4.

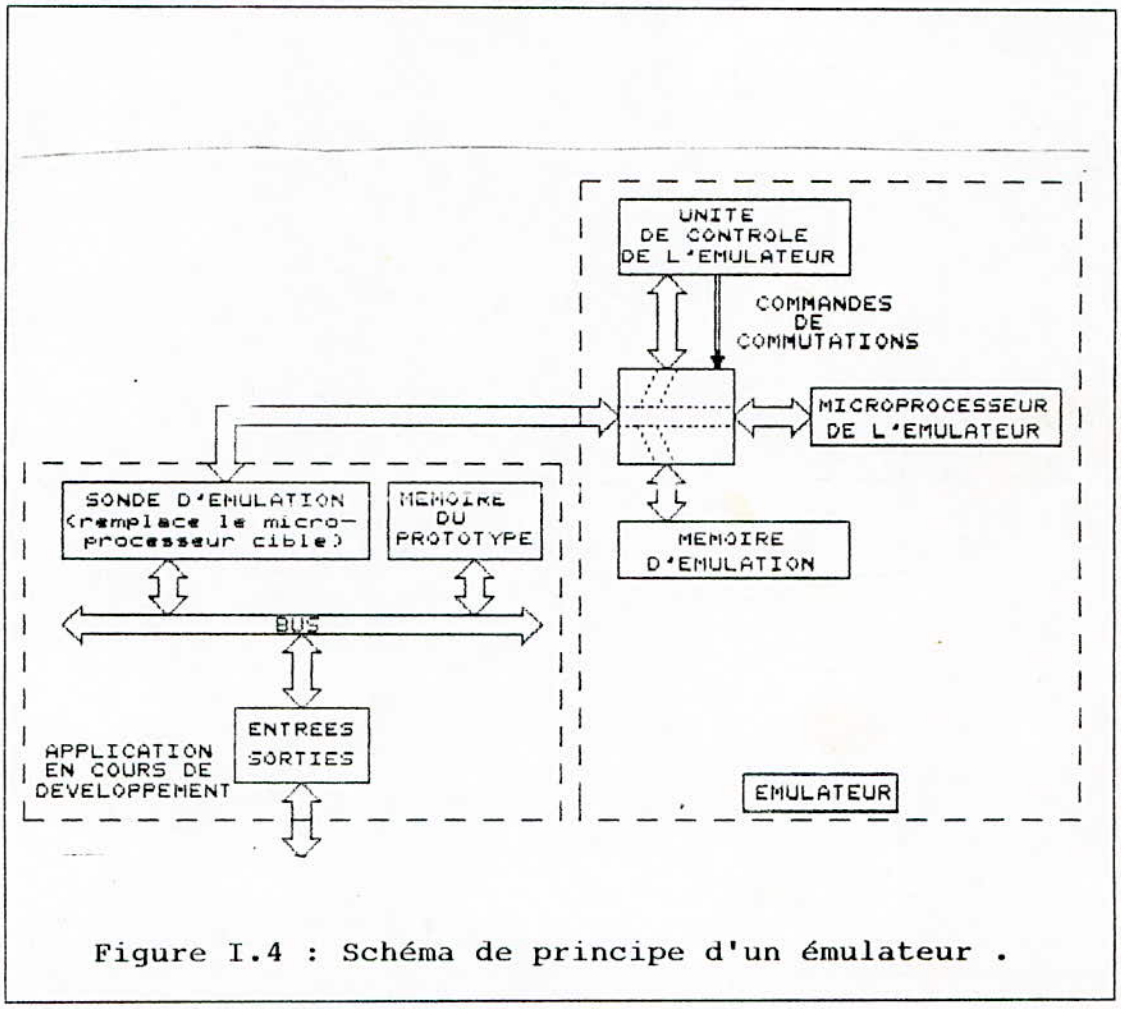


Figure I.4 : Schéma de principe d'un émulateur .

L'unité de contrôle permet de gerer les différents élements de l'émulateur (sonde, mémoire, microprocesseur, ...) et d'éxecuter les différentes commandes de la fonction d'émulation. En effet on pourra :

- charger un programme dans la mémoire d'émulation (à partir du système de développement);

- lire ou écrire dans un registre du microproceseur de l'émulateur;

- lire un programme de la mémoire d'émulation;

- éxecuter un programme.

Les opérations mémoires peuvent faire appel soit à la mémoire d'émulation,soit à la mémoire du prototype.A l'émulateur on associe,souvent,un analyseur logique qui permet de visualiser,en temps réel, l'ensemble des signaux du microprocesseur (bus de données, d'adresses et de commande).

L'émulateur est la partie la plus importante dans tout système développement. On verra, plus en détails, la fonction d'émulation dans le chapitre qui suit.

CHAPITRE II :

QU'EST-CE QUE L'EMULATION?

Lors du développement d'un prototype à base de microprocesseur, un certain nombre de tests importants précede l'émulation. Cette dernière n'intervient qu'une fois le logiciel et le materiel, testés séparèment, fonctionnent correctement.

1.DEFINITION DE L'EMULATION :

L'EMULATION, action d'émuler, est la "simulation" materielle du microprocesseur ou du microcontrôleur d'un prototype. En d'autres termes l'émulation est la substitution du µP de l'application par un système ayant un comportement identique mais contrôlable. L'émulation consiste à charger dans le système de développement le programme tel qu'il devrait etre dans le produit final. Le lancement de l'exécution de ce programme à partir du SD permet de piloter l'ensemble du prototype. L'émulation est la partie la plus importante dans un SD.

L'émulation, donc, n'est pas une simulation au sens habituel. Cette dernière est également utilisée dans le développement de produits.

LA SIMULATION : Dans ce cas, l'ensemble ou une partie des fonctionnalités du produit à réaliser est confié au SD. Les fonctionnalités logicielles peuvent etre réalisées et testées sur le SD.

Par contre, les fonctionnalités materielles ne peuvent etre que simulées , en totalité si elles sont simples, ou en partie dans le cas où elles sont complexes (opération de simulation difficile ou temps de simulation important). Dans ce dernier cas, la solution est de materialiser ces fonctions (hard) sur une carte électronique à wrapper (disponible sur la majorité des systèmes de développement) qui simule partiellement ou totalement la tache materielle attendue par le produit final. La simulation offre comme avantages une maitrise de la faisabilité et la possibilité de simuler un microprocesseur qui n'est pas encore disponible. Néanmoins, aucune maitrise des problèmes d'environnement dans lequel devra fonctionner le produit. D'où la nécessité de l'émulation.

REMARQUE :

Avant de passer à l'opération d'émulation, il est important de la situer parmi les autres opérations l'accompagnant. Entre autres les differents tests et vérifications, objet du sous- chapitre suivant.

2. TESTS :

2.1. Tests materiels:

2.1.1. "Smoke test :

Le premier test à effectuer lors du développement d'une carte est un test appelé "SMOKE TEST". Ce test consiste à enlever tous les boitiers (chips) et en branchant ensuite l'alimentation. Ce test permet de localiser d'éventuels court-circuits et de vérifier le bon branchement de l'alimentation.

2.1.2. Test des EPROMs:

L'un des problemes souvent rencontré est celui de la lecture des EPROMs. Il faut alors executer un test qui vérifie la bonne lecture de celles-ci. Pour ce faire, il faut charger des valeurs particulières dans les EPROMs de la carte. Ces valeurs sont 00H, FFH, AAH et 55H. Ces deux dernières valeurs servent, surtout, à la vérification de l'interaction des lignes de données. Deux types d'erreurs peuvent surgir :

Type 1 : Liées à l'environnement direct des EPROMs

- décodage d'adresse défectueux;
- problème au niveau de la selection des boitiers EPROMs, du bus d'adresses et/ou de données et des signaux de lecture et/ou d'écriture;
- et le temps d'accés...etc

Type 2 : L'erreur peut provenir d'une mauvaise connexion de la sonde d'émulation ou de celle de l'analyseur logique sur le proto type.

Elle peut etre, aussi, dûe à une mauvaise implantation des bus d'adresse, de donnée et de commande, ou à un problème d'horloge...etc

2.1.3. Test des RAMs:

Ce test se fait par l'écriture des valeurs 00H, FFH, AAH, 55H puis par la lecture et la vérification aprés chaque écriture. S'il y a disfonctionnement à ce niveau, cela peut etre dû à un mauvais décodage d'adresse (chip-select), mauvais branchement des bus adresse et/ou donnée. Elle peut aussi provenir des signaux de lecture/écriture et du temps d'accés.

2.1.4. Test des circuits d'interface périphériques :

a) Interfaces d'E/S parallèles programmables :(ex:PPI,PIA)

Une methode de tester ces interfaces est d'écrire en entrée puis en sortie et de vérifier la cohérence avec l'état physique des E/S, mesurer les tensions et voir si les niveaux des tensions correspondent aux niveaux prévûs.

b) Interface d'E/S série : (ex: UART,USART,ACIA)

Le test de ces circuits se fait en imposant des valeurs sur les lignes d' entrées parallèles (au moyen de microswitchs et de résistances) et effectuer ensuite des écritures vers un terminal (émission). Pour tester la reception de données (une fois le test de l'écriture fait et positif), il suffit d'écrire un petit programme de lecture/écriture, qui renvoie la donnée reçue du clavier vers l'écran. Il suffirait alors de taper une touche du clavier et d'attendre l'affichage sur l'écran le caractère correspondant.

Pour ces derniers et les autres interfaces (contrôleur d'écran, disque...etc) des tests spécifiques à chacun d'entre-eux devront etre mis au point suivant les fonctions auquelles ces interfaces sont dédiés.

c) Test des timers programmables:

Il suffit de les programmer avec des valeurs différentes et d'essayer de visualiser les fréquences issues à l'aide d'un oscilloscope ou de les mesurer à l'aide d'un compteur.

d) Convertisseurs A/N et N/A :

Mettre au point un programme d'acquisition de données et utiliser en sortie ou en entrée un scope ou un multimètre pour y vérifier la cohérence des données et proceder aux corrections éventuelles (potentiomètres ou autres). Les sources d'érreurs possibles sont :

- décodage d'adresse;
- temps d'accés;
- arrivée des signaux de lecture et écriture;
- gestion des interruptions;
- valeurs des composants passifs (condensateurs,résistances) inadéquates;

2.1.5. Test materiel global :

Si les tests précédents ont été effectués sur chacun des modules et s'avèrent positifs, il restera à executer un test de l'ensemble du materiel. En cas de non fonctionnement, une source d'erreur possible est au niveau des drivers c'est à dire au niveau des signaux de commande.Les problèmes, souvent, rencontrés lors d'un disfonctionnement partiel ou total sont dûs à l'interconnexion des différents modules composant la carte.

Exemple: Module d'interface UART pour une liaison terminal/unité centrale (UART composant relativement lent par rapport au microprocesseur). Le composant le plus lent impose sa vitesse. De ce fait il faudra retarder les signaux de lecture/écriture de l'UART jusqu'à sa disponibilité.

2.1.6. Test "Burn In":

Il est conseillé de faire des test dits "Burn In" qui consiste à mettre sous tension l'ensemble des cartes pendant une période de 24 à 48 heures. Ce test permet de vérifier la fiabilité du produit et la consommation des boitiers. Ce test est éxecuté en implantant un simple programme d'affichage continu d'un message sur l'écran.

2.2. Tests logiciels :

Pour les tests logiciels ci-dessous, l'outil utilisé est le de bugger implanté sur les systèmes de développement et sur les compatibles (exemple: DEBUG, SST, TURBODEBUGGER).

2.2.1. Tests unitaires statiques:

Ils consistent à tester chacune des procédures des differents modules, en simulant des entrées/sorties et en affectant des valeurs aux différents paramètres d'une procédure. Ces valeurs affichées et imprimées pour optimiser le programme ou localiser des erreurs.Il recommandé de tester les valeurs limites ("effets de bord").

2.2.2. Tests modulaires statiques:

Ces tests se situent à un niveau de modularité supérieur. Ils consistent à tester une fonction proprement dite (composée d'un ensemble de procedures) en simulant un certain nombre de données ou de paramètres.

2.2.3. Tests globaux semi-dynamiques :

Une fois les modules, testés individuellement, fonctionnent normalement, on procède aux tests globaux semi-dynamiques. Ces tests consistent à assembler tous les modules, préalablement vérifiés, et à faire des tests globaux en simulant les parties materielles soit par logiciel soit par des accés à des parties materielles disponibles sur le système développement et accessibles à l'utilisateur. Ces tests sont dits semi-dynamiques car ils correspondent aux différents cas de fonctionnement indépendemment du materiel, celui-ci étant simulé. A la localisation d'une erreur les tests logiciels effectués précedement devront etre refaits. Cette manière de faire est lassante, certes, mais néanmoins offre un gain de temps considérable.

Commandes disponibles sur un DEBUGGER :

→ Commande d'exécution de programme:(GO)

La commande GO transfert le contrôle au programme en cours de test.Deux types d'exécution sont possibles :

- Exécution continue GO FOREVER
- Exécution avec point d'arrêts GO FROM address TILL break point

L'exécution du programme de test lancée par la première commande n'est arrêtée que si une erreur liée au système d'exploitation du SD ou du PC survienne ou bien par sollicitation d'une touche d'arrêt prédinie (exemple: Contrôle X, ou contrôle break,ou autres touches). La deuxième option permet le lancement de l'execution du programme à partir de l'adresse spécifiée jusqu'au point d'arrêt mentionné.

→ Mise en place des points d'arrêts :

Un mode d'exécution possible offert par un debugger est celui permettant des points d'arrêts (break points). Cette commande est utilisée par la commande précedente GO FROM address TILL break point.

→ Execution en pas à pas :

Cette commande (dont le nom est souvent STEP) est utilisée pour éxecuter le programme instruction par instruction, en visualisant les registres internes du processeur et/ou des zones mémoires pour detecter une quelconque anomalie des valeurs attendûes.

→ Visualisation et modification des valeurs des registres.

→ Visualisation et modification du contenu de la mémoire.

→ Visualisation et modification du contenu des ports d'E/S.

→ Désassemblage :

Cette commande permet de visualiser une partie ou l'ensemble du programme exécutable (code binaire) sous forme mnémonique (language assembleur).

3. L'EMULATION :

Dans les tests précedents le logiciel de l'utilisateur est testés dans un materiel qui n'est pas celui auquel il a été destiné (car exécuté sur le materiel du SD). Idem pour le materiel conçû , il a été testé indépendamment de son logiciel. Le principe de l'étape suivante est de tester l'ensemble du logiciel dans son environnement véritable. C'est l'opération d'intégration du logiciel dans le materiel. On effectue des tests grâce :

→ à un outil materiel : la SONDE d'EMULATION;

et

→ à un outil logiciel : la FONCTION d'EMULATION.

3.1. MISE EN PLACE DE LA SONDE D'EMULATION :

L'ensemble du prototype et le système de développement doit etre hors tension. On enlève le microprocesseur du prototype et on le substitue par le pode de la sonde d'émulation. Le positionnement du connecteur devra correspondre au brochage du support du microproceseur de la carte prototype. La sonde est spécifique à chaque type de microprocesseur.Voir figure II.1

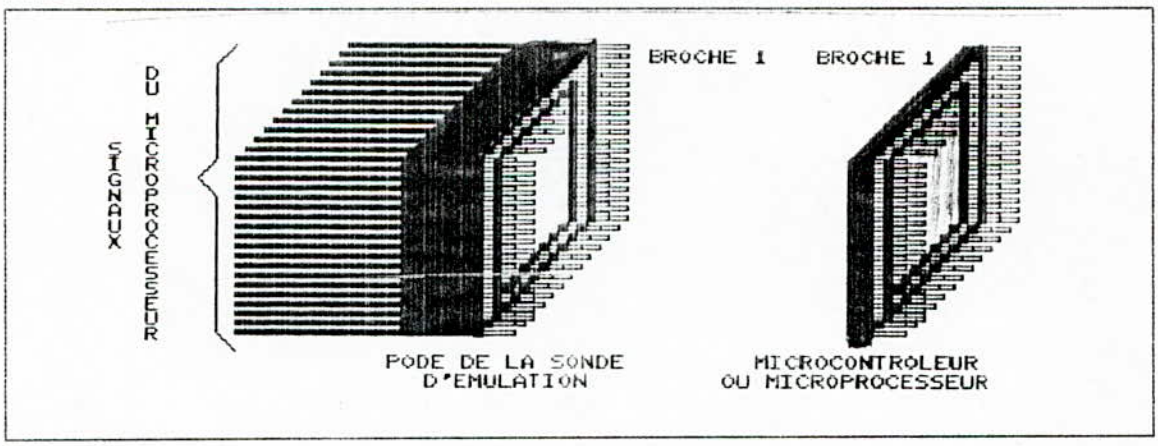


Figure II.1 : PODE de la sonde d'émulation.

Les fonctions de chaque broche du microprocesseur doivent se retrouver sur la broche correspondante du pode de la sonde d'émulation.Voir schéma II.2

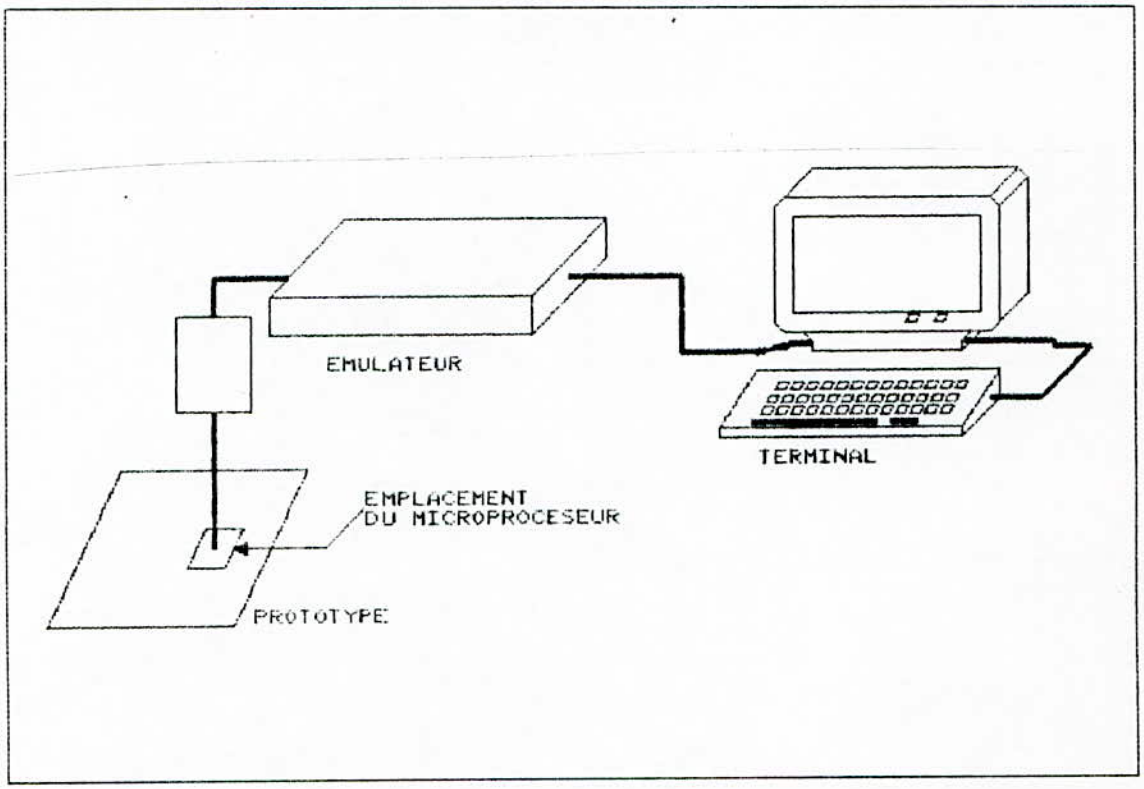


Figure II.2 : Emplacement de la sonde d'émulation.

Les SD qui émulent plusieurs microprocesseurs doivent avoir plusieurs sondes d'émulation.

3.2. TEST ET EMULATION :

3.2.1. Tests dynamiques:

Grâce à la fonction d'émulation, l'opérateur peut tester le logiciel dans son environnement réel et déceler également toute erreur matérielle. Les tests dynamiques englobent :

→ l'ensemble des fonctions du prototype dans les différents cas de figure;

→ l'interaction entre ces fonctions;

→ l'ensembles des signaux d'E/S du materiel et des logiciels de pilotage et des traitements de ces signaux;
→ les interruptions;
→ L'ensemble des phénomènes temps réel;
→ ...etc

3.2.2.Déscription des commandes d'un émulateur:

Ci-aprés seront décrites les différentes commandes disponibles dans un émulateur des plus performants. Cependant on peut en trouver des moins performants offrant un minimum de service (les commandes les plus importante) au développeur modeste.

→ Initialisation :

L'initialisation se fait par chargement du programme d'émulation, et par la selection du type d'horloge interne ou externe.

→ Aide :

Cette commande permet de donner la signification des commandes disponibles sur l'émulateur.

→ Configuration mémoire :

Trois configurations possibles pour la mémoire :

- mode partagé :la mémoire est partagé par le système de développement et le prototype;
- mode utilisateur : le programme utilisateur sera chargé dans la mémoire utilisateur du prototype;
- mode accés interdit : dans ce mode l'utilisateur désigne des zones mémoires inaccessibles ou bien le système de développement interdit l'accés à certaines zones jugées importantes.

Dans le mode partagé l'utilisateur définie des zones partagées par le programme d'émulation, les ressources du système de développement auquelles l'émulateur fait appel et le programme utilisateur.

→ Chargement en mémoire du programme utilisateur :

Une commande de chargement des programmes utilisateurs est d'une grande importance. Elle souvent nommée "LOAD".

→ Archivage :

Cette commande sert à imprimer l'ensemble des informations visualisées sur l'écran du système développement.

→ Affichages de table des symboles :

Cette commande permet de suivre l'évolution des variables (symboles) du programme utilisateur. En d'autre termes, cette commande offre la possibilité d'afficher le contenu de la table des symboles.

→ Definition du type de code d'affichage des valeurs numériques:

Cette commande donne la possibilité d'afficher des valeurs numériques dans une base selectionnée (binaire, octale, décimale, héxadécimale,..).

→ Lancement de l'émulation :

Le lancement de l'émulation peut se faire d'une manière continue (FOREVER) ou sous forme conditionnelle (TILL). Dans ce cas, continu l'émulation commence et aucune condition ne peut l'arrêter excèpté l'activation d'une touche du clavier prédifinie pour cet effet (exemple: touche "ESC" ou "END"). Au cours de l'émulation, il arrive que le système se plante à cause d'une erreur fatale touchant le contexte utilisateur ou surtout celui de l'émulateur. Il faudra, alors, réinitialiser le système et recharger le programme utilisateur.

→ Emulation en pas à pas :

A chaque exécution d'une instruction les registres du microprocesseur sont affichés : c'est l'émulation pas à pas logicielle.

La syntaxe habituelle de cette commande est :

STEP adresse FROM adresse
COUNT n
TILL expression conditionnelle
FROM adresse : adresse de départ chargé dans le compteur du programme;
COUNT n : nombre de pas (pas d'une, deux instructions);
TILL exp.cond. : expression logique composée de relations.

L'expression conditionnelle peut etre sous la forme d'une variable suivie d'un registre (ex:octet, mot, port, registre,..) et suivi d'une relation de type =,<,>,<=,>=,...etc.

→ Visualisation du bus microprocesseur :

Cette commande, souvent appelée TRACE, permet de visualiser l'état du bus microprocesseur : c'est l'émulation pas à pas materielle. Cette commande utilise un buffer dit "buffer trace" qui contient toutes les informations d'état du microprocesseur. Il constitue, en fait, un historique de l'évolution de l'état du bus du microproceseur. Il contient un certain nombre de frames (jusqu'à 1023 frames dans certains émulateurs) ou "d'images". Certains système de développement dispose de sondes externes pour analyser les signaux du microprocesseur (dans le cas où l'émulateur ne dispose de "buffer trace") ou de ses périphériques.

→ Désassemblage :

Cette commande permet de désassembler une partie ou l'ensemble du programme. En d'autres termes, elle permet d'afficher le programme exécutable (code binaire) sous forme de mnémoniques.

→ Modification des instructions du programme à tester : L'émulateur offre à l'utilisateur la possibilité de modifier son programme.

→ Validation et inhibition de signaux ou fonctions:

L'émulateur permet de valider ou d'inhiber d'une manière permanante ou temporaire des signaux ou des fonctions.

→ Définition et suppression de symboles :

Cette commande donne à l'utilisateur la possibilité de définir ou de supprimer des synboles.

→ Définition et modification du contenu des registres internes du microprocesseur :

Cette commande permet de visualiser le contenu des registres et de pouvoir les modifier.

→ Visualisation et modification de la mémoire:

Possibilité d'afficher la mémoire et de la modifier.

→ Visualisation et modification des ports d'E/S:

Commande permettant d'affecter des valeurs à des ports d'E/S.

→ Modification du contenu d'une adresse :

Cette commande permet d'affecter une valeur à un symbole représenté par une adresse.

→ Structure d'une suite de commandes :

Elle offre la possibilité de créer une suite de commandes plus particulièrement pour l'émulation des logiciels temps réel. Dans ce cas, les phenomènes sont rapides de sorte que le temps de taper la commande voulûe est trop important.

→ Macrocommande :

Offre la possibilité de définir une macro-commande à partrir de l'émulateur

exemple :

```
DEFINE MACRO AFFICHE
REGISTER
PORT 52
ENDM
```

La macro-commande AFFICHE permettra d'afficher tous les registres du microprocesseur et le contenu du port 52H.

→ Sortie de l'émulateur :

Souvent appelée EXIT permet de sortir du mode d'émulation et retour au système.

3.2.3. Les points d'arrêts materiels :

Similaires aux points d'arrêts logiciels, ils ne diffèrent que par la condition d'arrêt qui est liée à des signaux électriques plûtot qu'à des états logiques d'un ou plusierus registres.

Les conditions d'arrêts peuvent etre :

- le contenu d'une adresse, en chargeant une valeur limite ou par relation >= ou <= à une valeur;
- l'état d'une donnée ou d'une interruption;
- l'état d'un signal (READ, WRITE,..);

- l'état des signaux externes au bus du microproceseur. Ces signaux sont disponibles sur certaines sondes d'émulation. Voir figure II.3

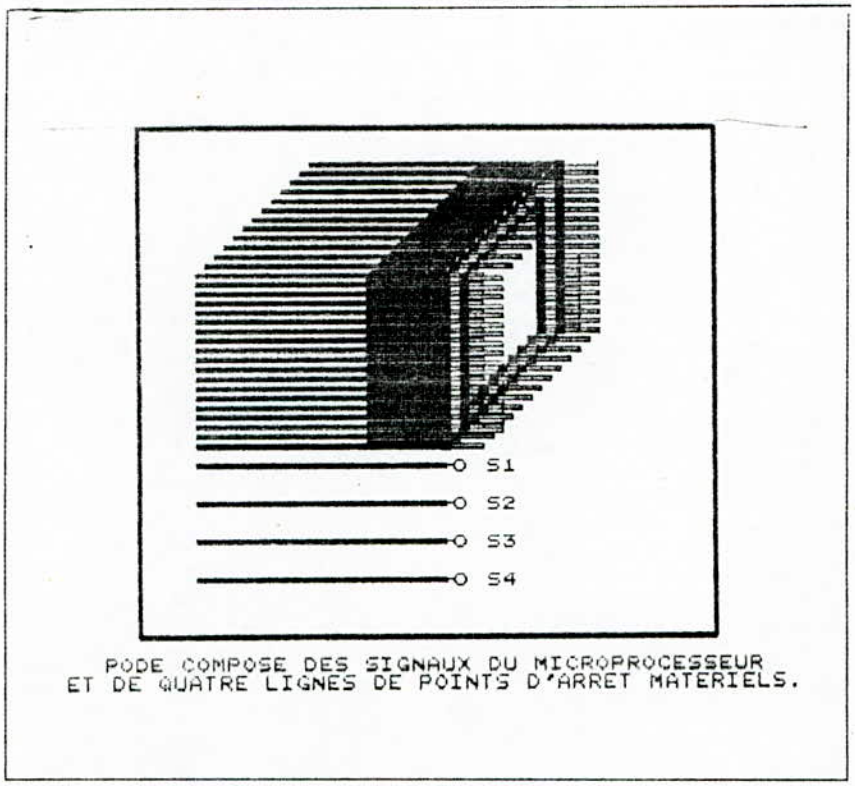


Figure II.3: Pode à signaux de points d'arrêts.

Les signaux S1,S2,S3 et S4 permettent des points d'arrêts materiels indépendants des signaux du bus du microprocesseur. La condition d'arrêt est composée des états des signaux S1, S2, S3 et S4 et des opérateurs logiques OR et AND.

On ne quittera pas ce chapitre sans avoir introduit un outil souvent associé à l'émulateur , l'ANALYSEUR LOGIQUE.

4. L'ANALYSEUR LOGIQUE :

L'analyseur logique est un outil de visualisation de signaux logiques (niveaux TTL, CMOS ..) à plusieurs trace. Il permet de suivre l'évolution des signaux des pines auxquelles sont branchées les fils de sa sonde. Il offre, ainsi, la possibillité de voir

la cohérence des signaux électroniques du prototype. Il est particulièrement efficace lorsqu'on rencontre des problèmes liés aux timing, aux délais de déclenchement et de synchronisation des différents circuits. Il permet, aussi, de déceler une anomalie de fonctionnement des circuits integrés en comparant les signaux visualisés à ceux prévûs (donnés par le constructeur). Aussi il rend possible la détection de déclenchements intempestifs, dus à des "glictchs" (impulsions positives ou négatives parasites,trés brèves). Ainsi, l'une des caractèristiques essentielle d'un analyseur logique est la possibilité de détecter des impulsions brèves de l'ordre de 3 à 4 nanosecondes et de les afficher. La puissance d'un analyseur logique réside dans l'importance de son utilisation avec le système d'émulation. En effet, plus cette interaction est grande plus le développeur a les moyens de localiser les erreurs rapidement et d'une manière efficace. Ceci permet entre autres de localiser l'origine d'un bug (logiciel ou materiel), d'évaluer l'interaction des instructions logicielles et des performances materielles.

CONSTITUTION D'UN ANALYSEUR LOGIQUE :

Un analyseur logique dispose :
- de plusieurs sondes qui se connectent par l'intermédiaire d'un jeu de cosses aux points de tests des bus ou des broches d'autres circuits;
- d'une entrée horloge faisant partie de l'ensemble des signaux d'entrées permettant à l'analyseur logique de fonctionner sur les mêmes impulsions d'horloge;
- d'une zone mémoire RAM pour sauvegarder les données acquises;
- de visualisation des données en mode état (héxedécimal, octal ou binaire) ou en mode temp (certains offrent même la possibilité de visualiser en mode mnémonique (assembleur);
- de deux délimiteurs de zones sur l'écran, ces deux délimiteurs permettent à l'utilisateur de mesurer les intervalles de temps

entre les signaux ou de délimiter une zone pour effectuer un zoom permettant une analyse plus fine.

Aprés l'étape d'émulation vient l'étape finale des tests, qui consiste à tester le produit indépendemment du système de développement (émulateur). C'est l'objet du prochain sous- chapitre.

5. Tests en " STAND ALONE " :

5.1. Tests sans émulateur :

Les tests sans émulateur consiste à tester le prototype dans son environnement réel. Les programmes sont chargés dans des EPROMs et/ou dans des RAMs. Les tâches auxquelles le produit a été cunçu devront etre testées une à une.

5.2. Test Burn In :

Les tests Burn In consistent à faire fonctionner le produit réalisé dans son environnement réel, sans liaison avec l'émulateur ou tout autre outil, pendant 48 heures.

5.3. Test utilisateur :

Ce sont les tests finaux liés à la souplesse d'utilisation pour un utilisateur non-informaticiens et non-électronicien. Ces tests concernent aussi les améliorations éventuelles à porter.

En guise de conclusion pour ce chapire, on peut dire que l'émulateur étant un outil puissant pour le développement, néanmoins, son association à un analyseur logique le rend sans égal en matière de développement.

CHAPITRE III :

PRESENTATION DE L'EMULATEUR REALISE "EMUL186".

Lors de la conception et la réalisation de la carte EMUL186 un certain nombre d'outils ont été mis à ma disposition entre autres:

- un analyseur logique à huit voies;
- une sonde logique;
- un oscilloscope;
- un multimètre;
- du materiel de wrapping
- programmateur d'EPROMS et de PAL;
- un PC compatible munis d'un certain nombre de logiciels de développement (DEBUG, SST, TURBOASSEMBLEUR ET TURBODEBUGGER);

En l'absence de système de développement, et plus précisemment d'émulateur, but de mon projet, la difficulté de ma tache s'est vûe accrûe. Ainsi, tout au long de la conception , on faisait recours à des solutions tantôt software tantôt hardware pour pallier à cette absence.

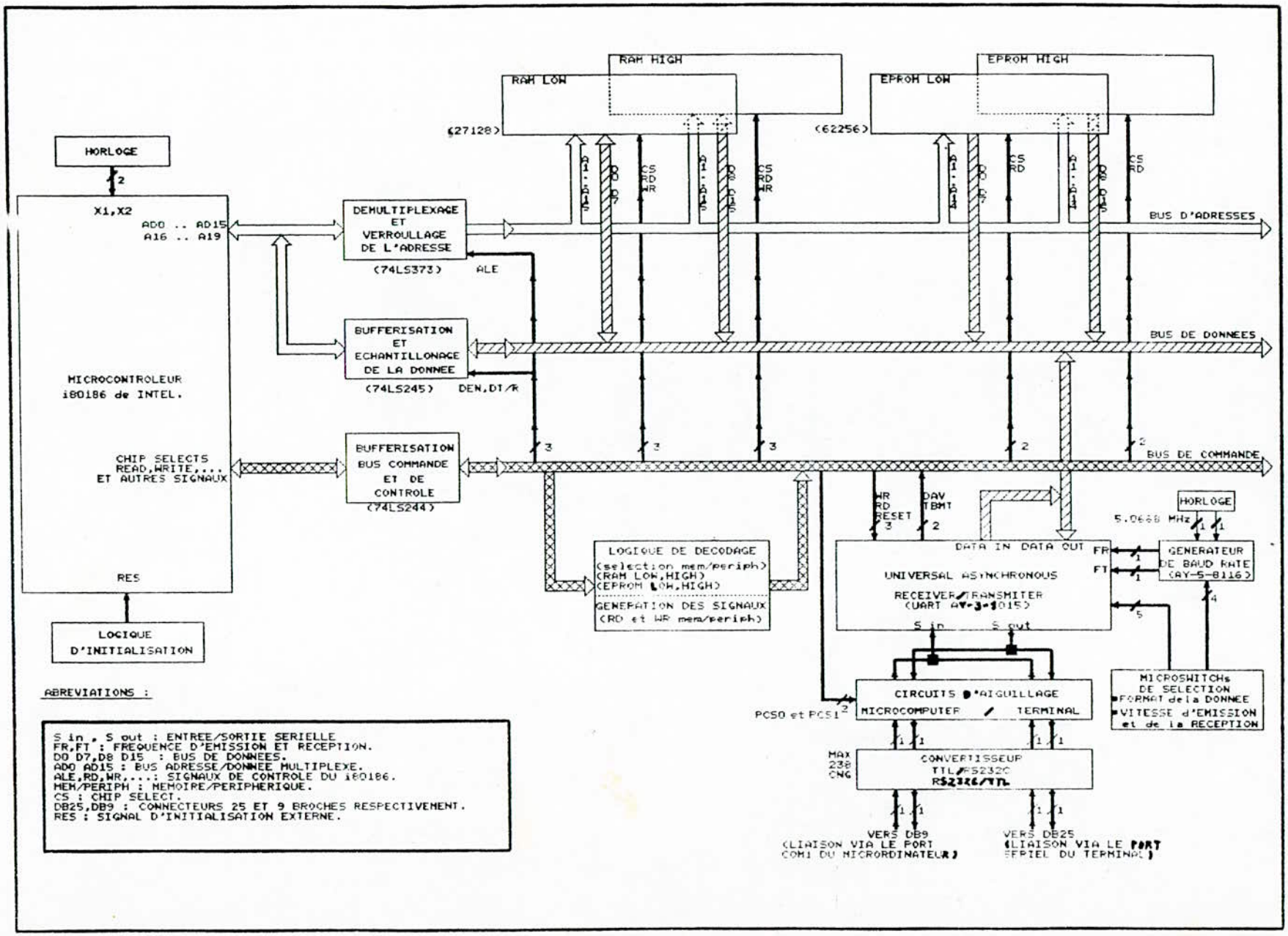
La carte réalisée, en wraping, contient un certain nombre de ressources, qu'on décrira tout au long de ce chapitre.

1.SYNOPTIQUE :

Ci-aprés est illustré le synoptique détaillé de l'émulateur réalisé, baptisé EMUL186, voir figure III.1.

Lors de sa conception, on avait tenu compte de la modularité fonctionnelle c'est à dire que toute la carte est divisée en bloc fonctionnels séparés. Tout cela afin de faciliter le debugging hardware c'est à dire de localiser une éventuelle erreur ou un aléa de fonctionnement. On présentera ci-aprés une étude détaillée de chaque module.

SYNOPTIQUE DETAILLE DE LA CARTE EMUL186 .



ABREVIATIONS :

S in , S out : ENTREE/SORTIE SERIELLE
FR,FT : FREQUENCE D'EMISSION ET RECEPTION.
D0 D7,D8 D15 : BUS DE DONNEES.
AD0 AD15 : BUS ADRESSE/DONNEE MULTIPLEXE.
ALE,RD,WR,...: SIGNAUX DE CONTROLE DU i80186.
MEM/PERIPH : MEMOIRE/PERIPHERIQUE.
CS : CHIP SELECT.
DB25,DB9 : CONNECTEURS 25 ET 9 BROCHES RESPECTIVEMENT.
RES : SIGNAL D'INITIALISATION EXTERNE.

2. PRESENTATION HARDWARE DU MICROCONTROLEUR 80186 :

Un microcontrôleur est un composant reprenant comme unité centrale un microprocesseur donné sur lequel on a implanté des mémoires RAM et/ou EPROM et des circuits périphériques tels que ports d'entrée/sortie, compteurs, temporisateurs, et convertisseurs (CAN,CNA)...etc. Le CPU du microcontrôleur 80186 partage la même architecture avec le 8086,8088 et le 80286. Il est complètement compatible, d'un point de vue logiciel, avec les microprocesseurs de la famille iAPX86, respectant la compatibilité ascendante qui donne à INTEL son hégémonie dans le domaine de la micro-informatique (domaine des PC compatibles).

CARACTERISTIQUES PRINCIPALES DU 80186 :

→ Il integre:

- le CPU (microprocesseur 8086);
- un générateur d'horloge;
- deux canaux DMA;
- un contrôleur d'interruption;
- trois timers programmables;
- une logique de selection boitiers memoire et ports d'E/S;
- un générateur d'états d'attente;
- un contrôleur de bus local;

→ Il existe en deux versions: 80186-10 fonctionne à 10 MHz et 80186 à 8MHz;

→ Il adresse directement un Mégabytes de mémoire et 64Kbytes d'E/S.

→ il est compatible d'un point de vu logiciel avec le 8086 et 8088. Son jeu d'intructions a été enrichi de 10 nouvelles instructions.

→ disponible en 68 broches sous trois formes: Voir annexe A.

- Plastic Lead Chip Carrier (PLCC):pour montage en surface sur les circuits imprimés;
- Ceramic Pin Grid Array (PGA):pour montage en insertion, avec des broches les quatre cotés et sous forme de deux rangées;
- Ceramic Leadless Chip Carrier (LCC),pour montage en surface.

SIGNAUX DU i80186 :

Les signaux du 80186 se subdivisent en plusieurs catégories, voir figure III.2.

Function	Signal name
Address/data	AD0-AD15
Address/status	A16/S3-A19/S6,BHE/S7
Co-processor control	TEST
Local bus arbitration	HOLD,HLDA
Local bus control	ALE,RD,,WR,DT/R,DEN
Multi-master bus	LOCK
Ready(w it) interface	SRDY,ARDY
Status information	S0-S2

Figure III.2 : Signaux du 80186.

BLOCS INTEGRES DU i80186:

A) CPU (Central Processing Unit):

Le CPU du i80186 est identique à celui du 8086,8088 et du 80286.

Il est totalement compatible d'un point de vue logiciel avec le 8086 et le 8088.

B)UNITE INTERFACE BUS (BIU):

Cette unité fournit les signaux de commande,de contrôle, d'état du bus et gère une file d'attente de 6 octets.

C)UNITE DMA (Direct Memory Access):

Le i80186 inclus 2 canaux DMA indépendants. Les canaux opèrent indépendemment du CPU et peuvent solliciter n'importe quel peri-phèrique integré (contrôleur d'interuption, chip-selects...) éxactement comme le CPU.

D) UNITE TIMER:

Le 80186 intègre une unité TIMER constituée de trois timers de 16 bits indépendants.Ces trois timers opèrent indépendemment du CPU. Deux,d'entre-eux,présentent une entrée et une sortie permettant le comptage d'évenements externes et la génération d'une forme d'onde carrée de fréquence quelconque.Le troisième timer peur etre utilisé en tant que tel, pour déclencher les autres timers ou comme demande DMA.

E) CONTROLEUR D'INTERUPTION INTEGRE:

Le contrôleur d'interuption permet la synchronisation des demandes d'interuption, l'attribution des priorités de ces demandes et la lecture de leur type en reponse à la prise en compte de celle-ci.

Il peut operer en deux modes principaux:

-non-iRMX mode(mode maitre);
-iRMX mode (mode esclave, dans le cas où un contrôleur d'interruption 8259 aurait été ajouté et configuré en mode maitre).

F) GENERATEUR D'HORLOGE:

Cette unité génère le signal d'horloge pour le CPU ainsi que pour tous les périphériques integrés du 80186. L'unité comprends aussi un circuit de réinitialisation et une logique d'états d'attente.

G) UNITE DES CHIP SELECT:(lignes de selection boitier)

Le 80186 un intègre aussi une unité de chip-selects programmable, qui génère des signaux de selection mémoire et entr.e/-sortie. Ces signaux sont : $\overline{UCS}$: Upper memory chip Select;
$\overline{LCS}$: Lower memory Chip Select;
$\overline{MCS0-3}$: Medium memory Chip selects;
$\overline{PCS0-6}$: Peripheral Chip selects.

Les lignes $\overline{UCS}$, $\overline{LCS}$ et $\overline{MCS0-3}$ partagent ainsi l'éspace mémoire en trois parties.Voir figure III.3

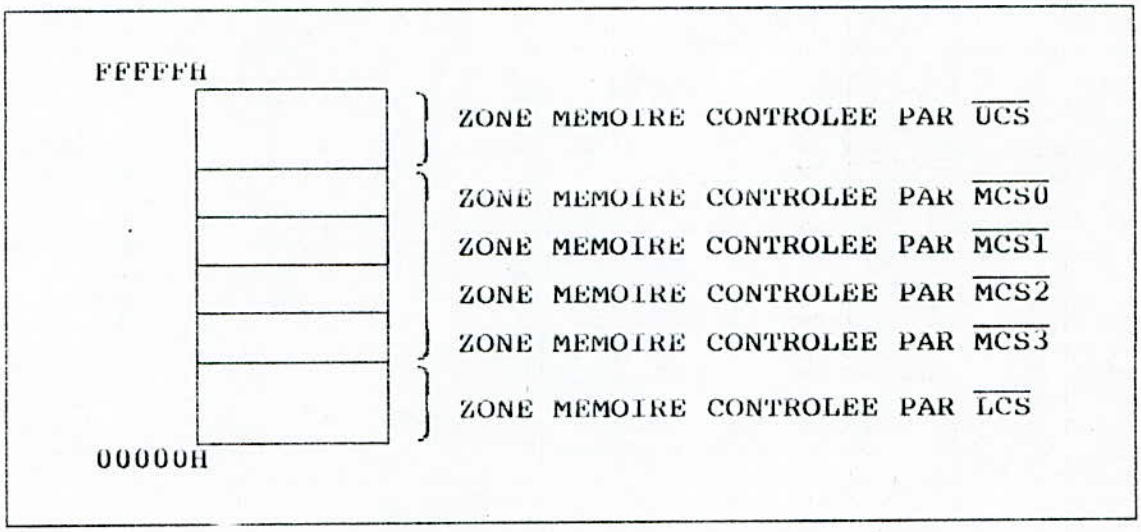


Figure III.3 : Zones mémoire et chip-selects.

Les lignes PCS0 à 6 sont valides pour sept bloc de 128 bytes dans l'esapce d'E/S.

La présentation succinte du hard 80186 s'arrête à ce niveau pour plus information se referer au datasheet de Intel "Microsystem components" iAPX186.

3.CIRCUITS D'INITIALISATION :

Pour réinitialiser le microcontrôleur, il suffit de mettre son entrée $\overline{RES}$ à l'état bas durant au moins quatre cycles d'horloge. Un circuit RC génére un signal exponentiel à l'allumage ou à l'enfoncement d'un bouton poussoir (permettant à l'utilisateur de générer une séquence d'initialisation manuelle). En effet, l'appui sur ce bouton décharge le condensateur et un autre cycle de charge, donc d'initialisation, est entamé. Une bascule RS

(74LS279), utilisée en anti-rebonds (élimination des rebonds du bouton poussoir, ainsi donc une bonne immunité au bruit). Le signal de sortie de l'anti-rebonds, associé à celui du circuit RC, attaquent une porte logique AND (74LS08). La sortie de celle-ci est reliée la pine $\overline{RES}$ du microcontrôleur. L'interface d'entrée/-sortie (UART AY-3-1015) nécessite un niveau haut pour son init.-lisation. On relie la pine XR à la sortie de la porte logique AND inversée.
(Voir schéma III.1).

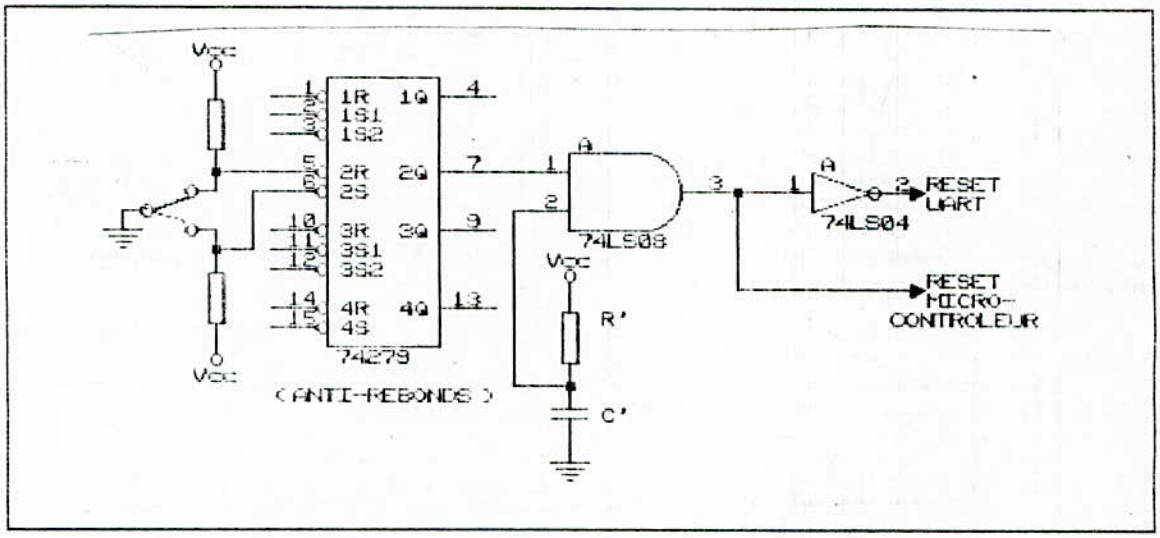


Schéma III.1 : Circuits d'initialisation.

Aprés cela, le 80186 commence l'exécution de l'instruction se trouvant à l'adresse FFFF0H.

4.CIRCUITS D'HORLOGE :

Le 80186 intègre un générateur d'horloge ne nécèssitant qu'un quartz externe pour générer le signal d'horloge.Voir schéma III.2.

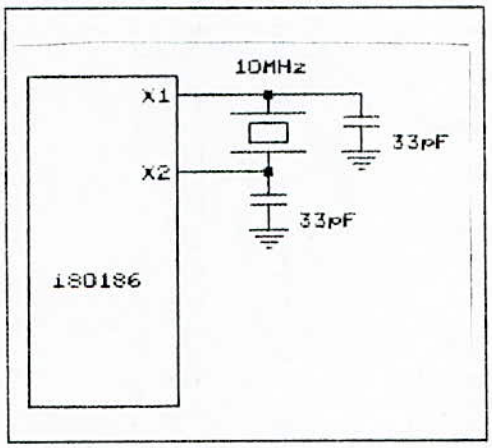


Schéma III.2 : Connexion du quartz.

5. LE BUS du i80186 :

Le 80186 possède un bus multipléxé adresse/donnée (AD0-AD15) de 16 lignes. Pour extraire la donnée et l'adresse, il faudra démultiplèxer ce bus. Le 80186 génere à cet effet un signal informatif de la présence sur le bus d'une adresse valide (ALE : Address Latch Enable) et d'un autre signal indiquant la présence d'une donnée valide ($\overline{DEN}$: Data ENable). Tenant compte du fait que les circuits périphériques (tels RAM, EPROM...) nécèssitent la présence d'une adresse stable lors de l'arrivée d'un signal de lecture ou d'écriture. Il faudra, alors, verouiller cette adresse. La sortance du 80186 étant de deux il est important de bufféríser le bus de données ainsi que celui de commande. Voir synoptique ci-dessus et schéma de la carte.

a) LE BUS D'ADRESSES:

Le 80186 peut adresser un mégaoctets de mémoire, il est important de verouiller les adresses dans un tampon. Le démultiplèxage n'est effectué que pour les 16 premières lignes AD0-AD15 par deux registres (huit bits chacun, 74LS373), un troisième registre est utilisé pour latcher les bits de poids fort A16-A19. L'ordre d'échantillonnage du bus est donné par le signal ALE (Address Latch Enable), indiquant qu'une adresse valide est présente sur le bus.

Voir schéma III.3.

b) LE BUS DE DONNEES :

Le bus de données est extrait du bus multiplèxé adresse/donnée du 80186, en utilisant deux buffers bidirectionnels de huit bits (74LS245). Le signal DT/$\overline{R}$ indique le sens de transfert des données (émission ou reception):

DT/R = 1 ⟶ transmission (émission):
microcontrôleur ⟶ environnement.

DT/R = 0 ⟶ reception :
microcontrôleur ⟵ environnement.

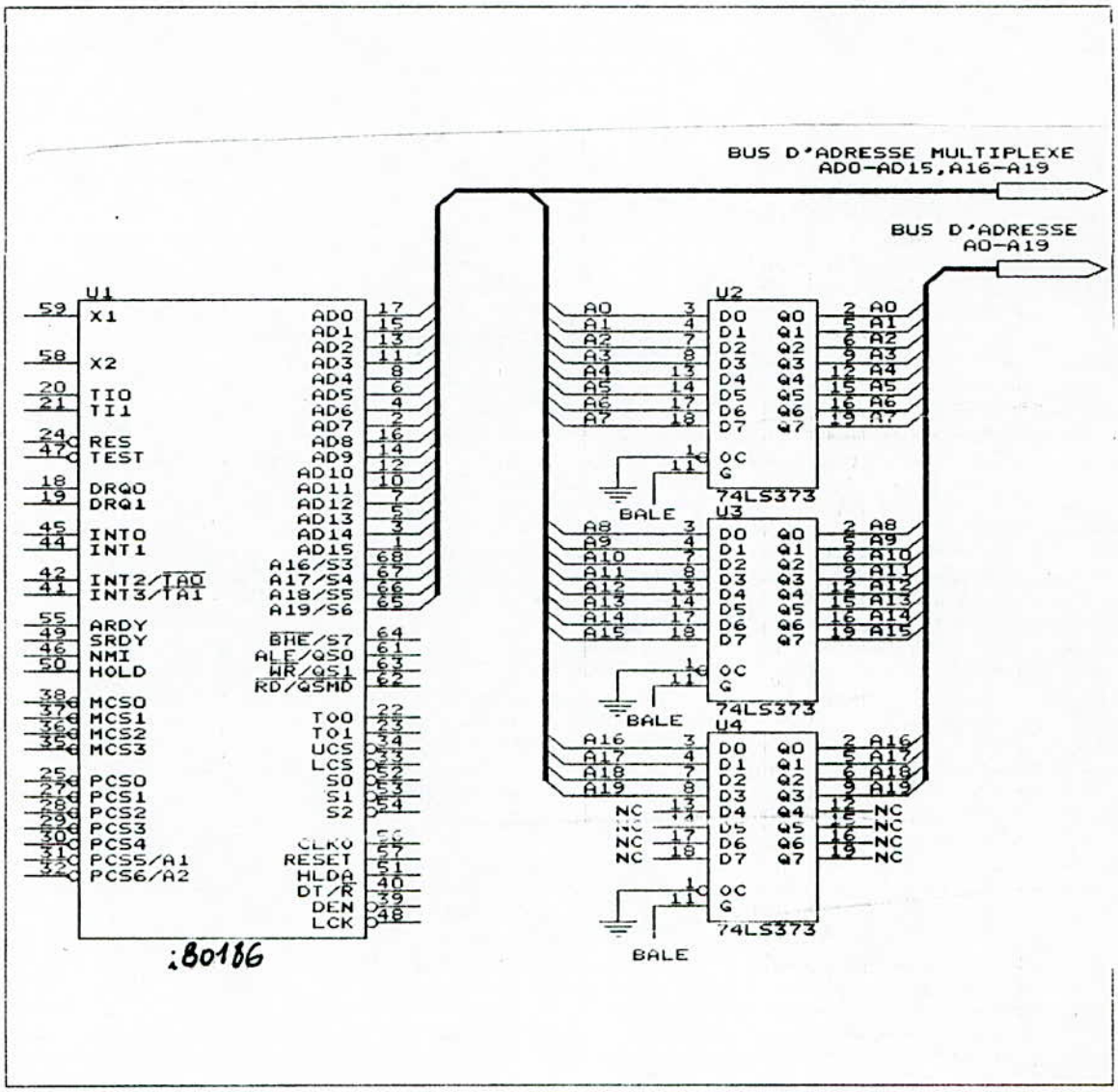


Schéma III.3 : Démultiplexage du bus.

Le signal $\overline{DEN}$ nous informe sur la présence d'une donnée valide et stable.
Voir schéma III.4.

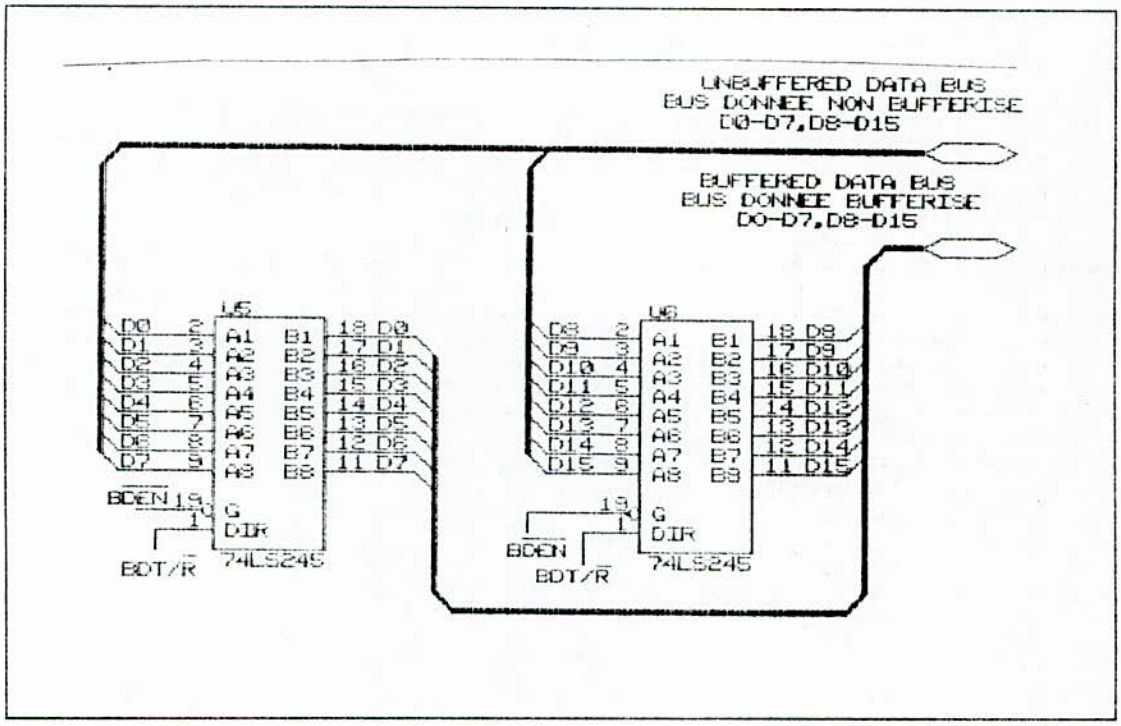


Schéma III.4 : Bufferisation du bus de données.

c)LE BUS DE COMMANDE :

Les sorties du 80186 peuvent attaquer une charge de 200 pF sans pertes de niveaux, soit donc une sortance de deux (le double de celle du 8086). Comme le nombre de circuits integrés à connecter au microcontrôleur peut s'averer important, on a pris, alors, la précaution de bufferiser ces signaux grâce aux buffers unidirectionnels (74LS244). Voir schéma III.5.

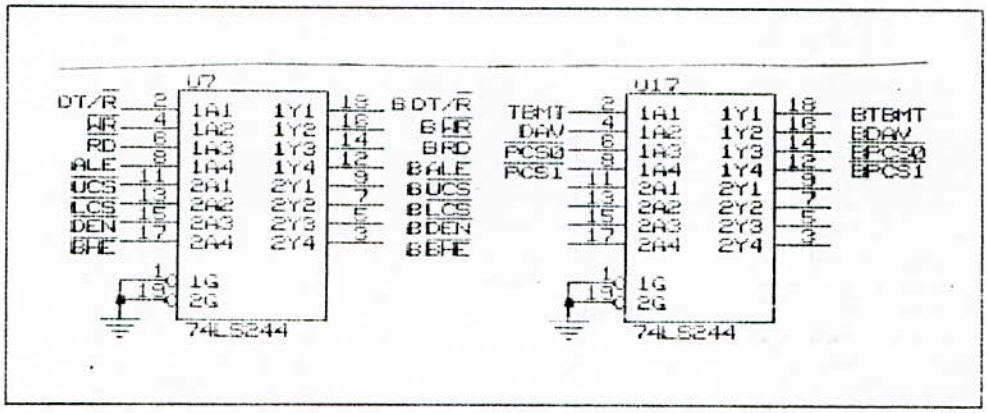


Schéma III.5 : Bufferisation du bus de commande et autres.

6. MODULE MEMOIRE :

Le microcontrôleur de INTEL i80186 peut adresser un mégabytes de mémoire soit deux bancs de 512 Kilobytes (banc poids faible,D0-D7, et banc poids fort, D8-D15). Cela est dû à l'absence, sur le marché, de mémoires dont le bus de données est de seize bits. Cette cision en deux bancs poids faible et poids fort permettra l'accés à l'octet ou au mot de seize bits (double octets). En effet, le i80186 effectue des opérations de lecture, d'écriture, arithmètiques et logique :

→ de mots de seize bits se trouvant à une adresse paire ou impaire;

→ de mots de huit bits de poids fort D8-D15;

→ de mots de huit bits de poids faible D0-D7.

Les lignes A0 (ligne d'adresse) et $\overline{BHE}$ (Bus High Enable) nous informe sur le type de traitement qui s'effectue (traitement en huit bits ou en seize bits). Pour le mégaoctets adressable (ou les 512 Kilomots), les octets de poids faible trouvent leur ligne d'adresse A0 à l'état bas, et à l'état haut pour les octets de poids fort. De ce fait, on peut donc utiliser la ligne A0 pour valider le banc de poids faible. Le signal $\overline{BHE}$ validera le banc mémoire de poids fort (Voir figure III.4). Il est, toutefois con

seillé de n'utiliser que des adresses paires, mais l'accès à un mot situé à une adresse impaire demeure possible mais quoique d'éxecution plus lente car le microcontrôleur l'executera en deux cycles (d'écriture ou de lecture).

BHE	A0	FONCTION
0	0	TRANSFERT D'UN MOT COMPLET.
0	1	// // OCTET D8-D15
1	0	// // OCTET D0-D7
1	1	RESERVE

Figure III.4 : SIGNIFICATION DES ETATS DE A0 ET $\overline{BHE}$.

Le contrôle des boitiers mémoire se fait par les signaux $\overline{RD}$, $\overline{WR}$, $\overline{UCS}$, $\overline{LCS}$, $\overline{MCS0-3}$.

MEMOIRES PROGRAMMES ⟶ EPROMs :

Deux EPROMS, poids faible et poids fort, ont été prévues pour le stockage des routines d'initialisation, du moniteur de gestion de l'émulateur et du debugger. Notons que la ligne A0 du bus d'adresses du microcontrôleur n'est pas reliée à la pine A0 du bus d'adresse des EPROMs. Du moment qu'on utilise DEUX EPROMs l'une poids faible l'autre poids fort. La pine A0 de l'EPROM est reliée à la pine A1 du bus d'adresse du microcontrôleur. La lecture des EPROMS se fait par mots de seize bits, ceci est le meilleur choix que l'on puisse faire. Sachant que la lecture par octet ne nous perdrait que du temps. Ce qui n'est pas pareil pour les RAMs, dont le mode d'accès est laissé au choix de l'utilisateur , du moment que l'accès à la zone comprenant le moniteur (EPROMs) devra lui est interdit.

Le 80186 ne fournit pas de signaux de lecture/écriture mémoire et entrée/sortie separés. De ce fait, il faudra les réaliser en combinantle signal $\overline{S2}$, les lignes de selection mémoire et E/S avec les lignes de lecture/écriture.

Pour la carte EMUL186 c'est la seconde solution qui a été retenue (voir schéma III.6) afin de garder les bits d'état pour le partage du bus avec d'autres ressources.

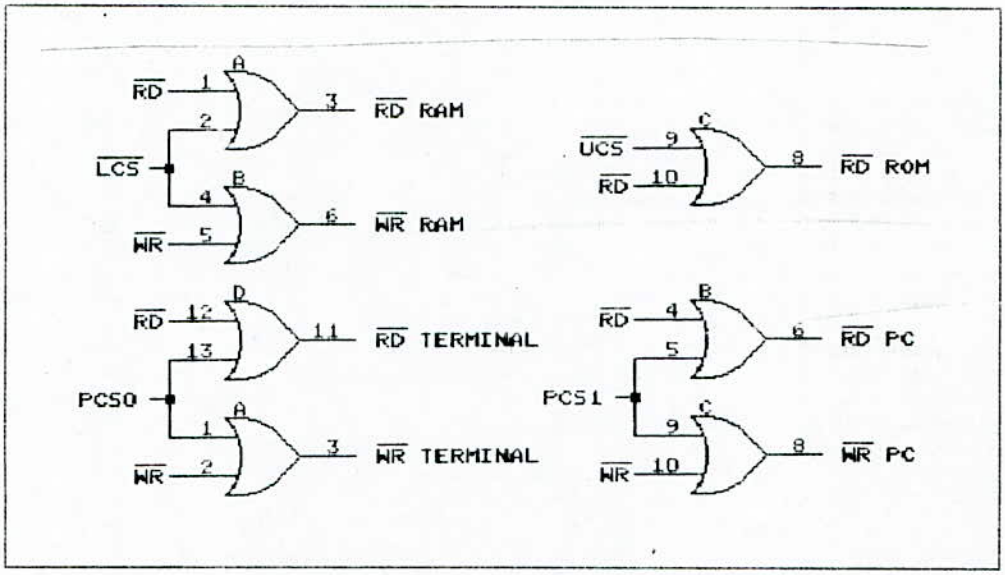


Schéma III.6 : Signaux lecture/écriture RAM,EPROM et E/S.

Le signal de validation du circuit (Chip Enable) $\overline{CE}$ devra preceder le signal de lecture. (Voir figure III.1)

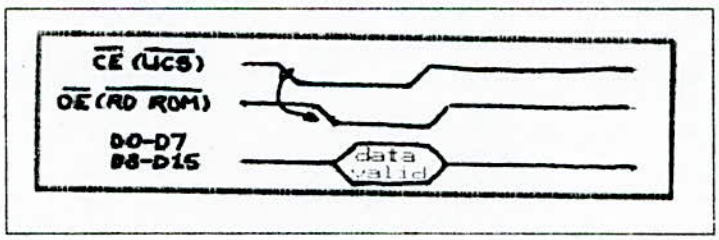


Schéma III.7 : Chronogrammes de lecture des EPROMs.

Cela se trouve satisfait du fait que le signal $\overline{BRD}$ ($\overline{RD}$ Bufferisé) passe par une porte logique OR dont la sortie ne passe à l'état bas que si $\overline{UCS}$ et $\overline{BRD}$ soient tous deux à l'état bas.

MEMOIRE DE DONNEES ⟶ RAMs :

Deux RAMs (de type 62256) ont été utilisées comme mémoire de données pour contenir les variables des programmes,ou tout simplement pour exécuter les programmes utilisateurs entrés à partir du clavier ou par téléchargement à partir d'un microrordinateur.
Le signal de lecture RAM est généré de la même façon que celui des EPROMs et est relié à la pine $\overline{OE}$ des boitiers RAM. Idem pour le signal d'écriture qui lui attaque la pine $\overline{WE}$ des RAMs (voir schéma III.6). La selection des bancs mémoire poids faible et poids
fort se fait par la combinaison du chip-select avec le signal A0 et BHE respectivement (voir schema III.8)

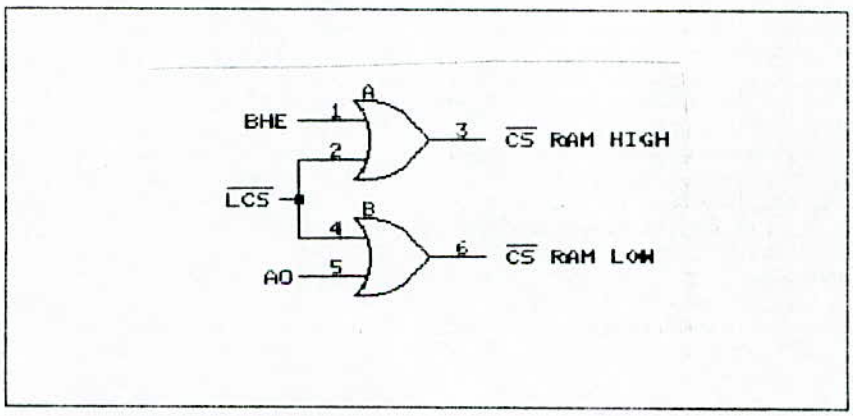


Schéma III.8 : Selection mémoire haute et basse.

Le schéma du bloc mémoire est donné ci-dessous:

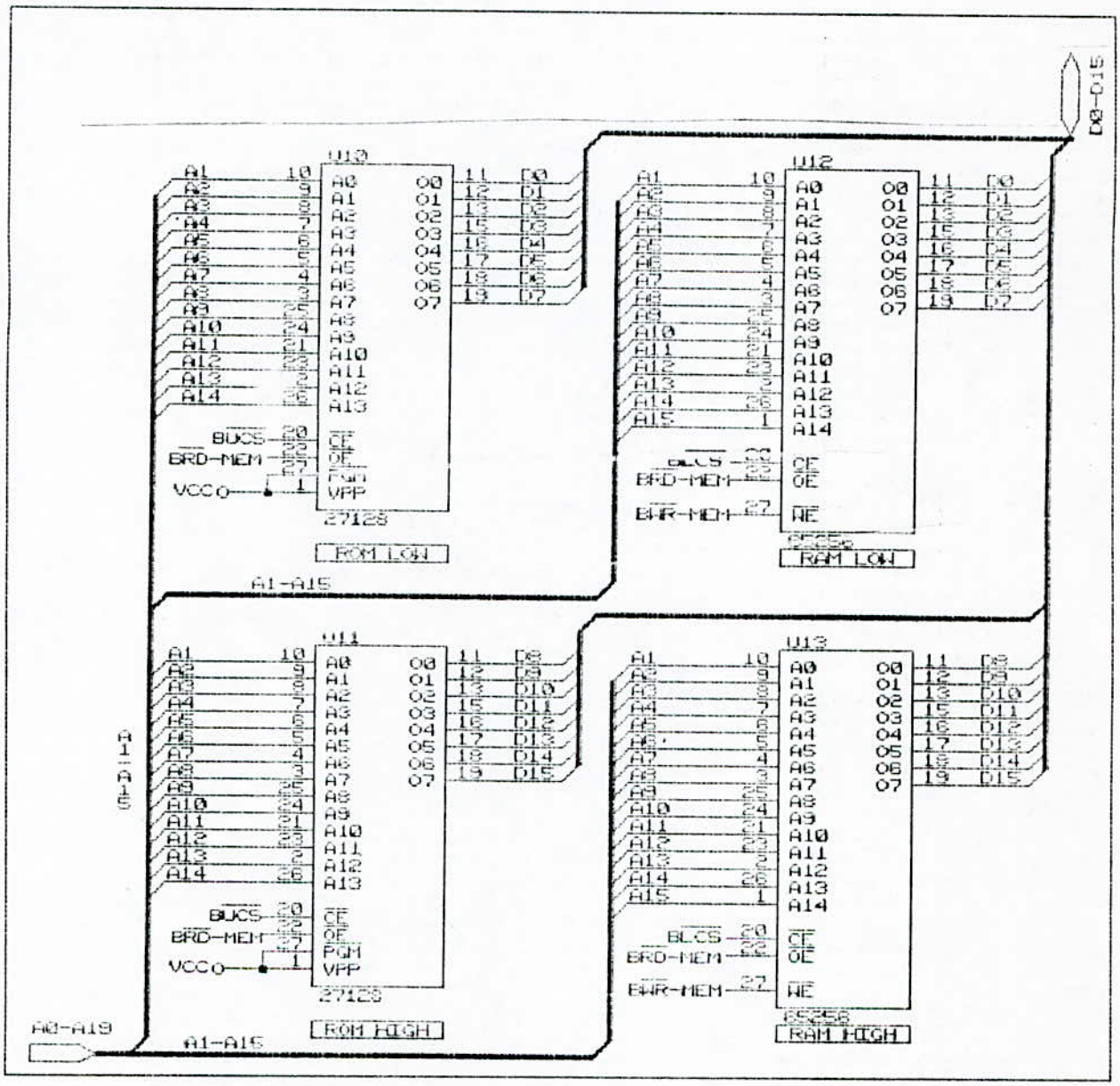


Schéma III.9 : Bloc mémoire.

7. L'INTERFACE D'ENTREE/SORTIE :

a) L'UART AY-3-1015 :

Pour la liaison EMUL186/TERMINAL ou EMUL186/PC on aura besoin d'un interface serie. Un interface serie est constitué d'un émetteur/recepteur asynchrone universel (UART) ou synchrone asycnhrone (USART) et d'un quelconque circuit de conversion de signaux compatibles TTL ("0" logique → 0 Volt, "1" logique → 5 Volts) en signaux compatibles RS232C ("0" logique → -12 Volts , "1" logique → +12 Volts) et vice versa. Le but de l'interface serie est de convertir des données parallèles en données seriélles (lors de l'émission) et inversement (lors de la reception). Quand un octet est émis vers l'UART, il est décalé bit à bit (au moyen d'un registre à décalage à entrées parallèles et à sortie seriélle unique) linéairement et transmis le long d'un fil unique au recepteur. Ce dernier parallèlise, de nouveau, ces données à son niveau et reconstitue ainsi la donnée.

CONFIGURATION DE L'UART:

L'UART devra etre configurée de sorte à ce que le format des données soit identique pour la carte EMUL186,le terminal et le PC:

FORMAT :

- → Nombre de bits de la donnée 5,6 ou 7;
- → Parité paire, impaire, ou sans parité;
- → Nomre de bits stop.
- → Vitesse d'émision des données;
- → Vitesse de reception des données.

(Pour la configuration de l'Uart et le GBR voir annexe A).

Voir schéma de configuration du format, ci-dessous.

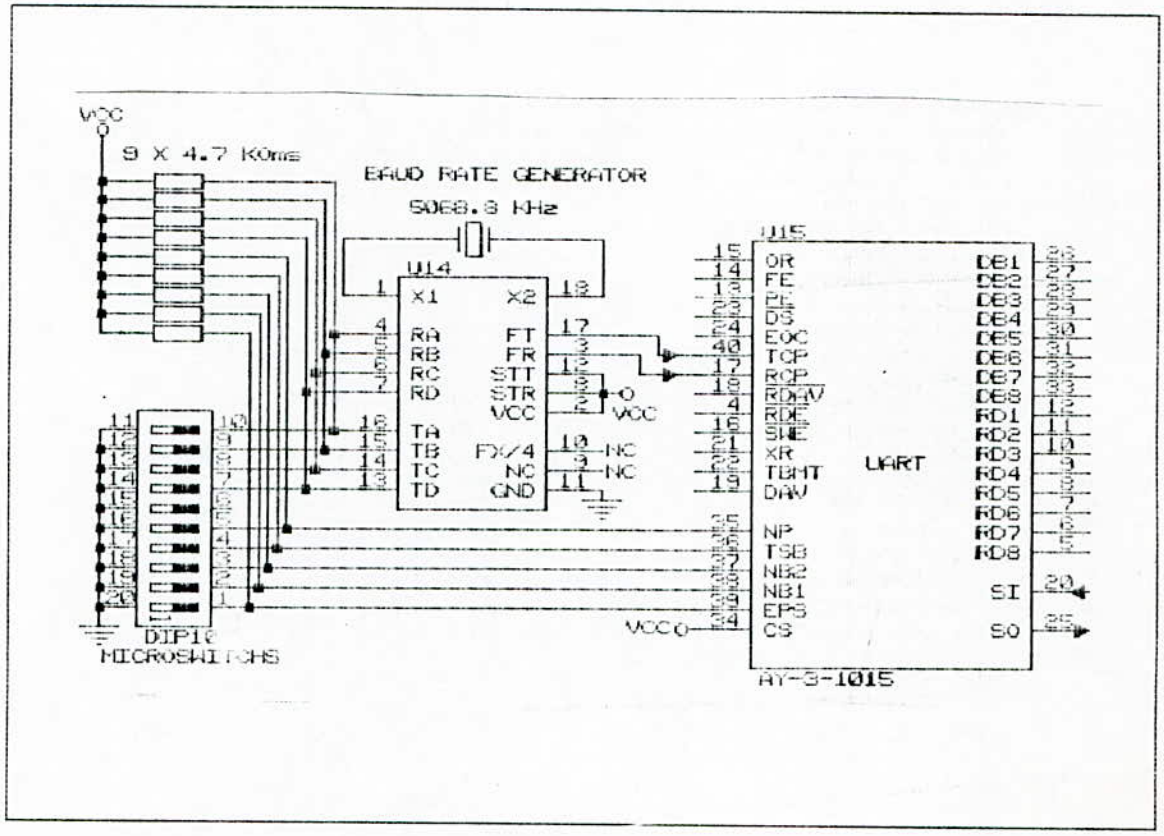


Schéma III.10 : Configuration de l'UART et du GBR.

L'UART est le circuit AY-3-1015. Cet UART est reliée à l'unité centrale par la ligne de lecture, la ligne d'écriture, une ligne d'interuption, le bus de données (D0-D7) et une ligne de mise en attente du processeur ,voir schéma de l'interface serie. Le format des données est choisie au moyen de microswitchs, la pine de validation du mot de commande (Command Strobe) est mise à "1" validant toujours le format, car on a utiliser une configuration par microswitchs, si s'était par programme on sera, alors, contraint de la reliér à une sortie d'une bascule D ou d'un latch contenant l'information de validation de format, aprés avoir positionner ce dernier. Les signaux $\overline{RD}$ et $\overline{WR}$ combinés avec PCS0 et PCS1 permettent l'autorisation d'envoyer/recevoir vers/du terminal ou du PC voir figure III.11.

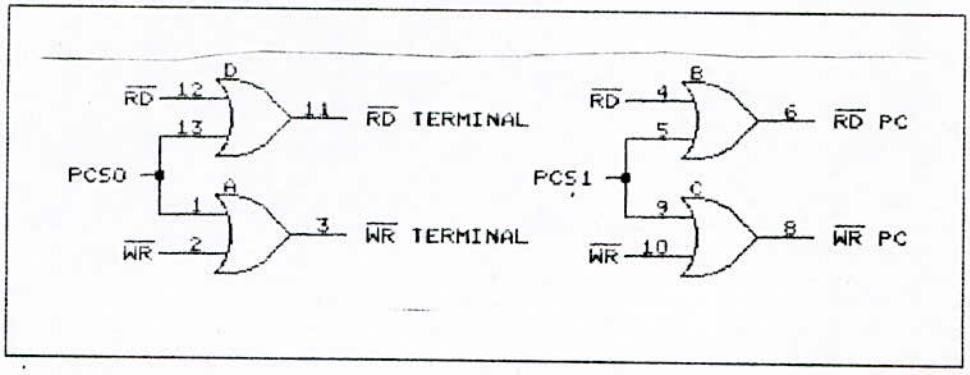


Schéma III.11 : Signaux de lecture/écriture d'E/S.

Le signal de lecture d'E/S attaque les pines $\overline{RDE}$ (Received Data Enable) et $\overline{RDAV}$ (Received DAta Valid). A l'appui sur une touche du clavier du terminal (si le mode de "downloading" ou de téléchargement n'est pas validé) la pine 19 de l'UART $\overline{DAV}$ (DAta Valid) bascule, celle-ci bufférisée attaque la pine INT0 du microcontrôleur, l'informant que le clavier a été sollicité. Ensuite le traitant de l'interruption décidera la prise en compte du caractère et de son interprétation. L'UART étant un composant relativement lent par rapport au processeur. Il est, alors, important d'inserer au cycle bus du processeur des états d'attente (en plus des états d'attente préprogrammés). Pour ce faire, un signal disponible sur l'UART informe le processeur continuellement sur l'état du bufter de transmision: c'est le signal TBMT. Ce signal est relié, aprés bufférisation à la pine ARDY du i80186. Cela permet d'éviter les problèmes d'écrasement des données.

b) La jonction RS232C :

Sur la carte , on a placé deux connecteurs l'un femelle DB25 (25 broches), l'autre mâle DB9 (9 broches) pour les liaisons emul186/ terminal et emul186/PC. Pour la transmission de données il faudra gerer un protocole de communication appelé HANDSHAKE.

On distingue deux sortes de handshake :

→ Hard handshake

→ Soft handshake

En effet, vû l'absence de système de developpement, on cherchait à simplifier au maximum le hard au dépend du soft (car il y a toujours un compromis entre le hard et le soft). C'est la raison pour laquelle on a opter pour le soft handshake.
Mais en fait en quoi réside la différence ? La réponse en est que l'émeteur ou le recpteur est informé de l'état de l'autre par des lignes materielles dans le cas du hard handshake et par des octets d'informations (XON/XOFF ou ACK/NAK ou autres..)dans le cas du soft handshake.

Des 25 signaux spécifiés par la norme RS232C seuls 3 sont utilisés pour la communication avec le terminal ou le PC:

→ TxData : émission de données ;

→ RxData : reception de données;

→ GND : masse logique (jointe à la masse mécanique).

Les signaux compatibles RS232C sont en logique ± 12 Volts. De ce fait ils faudra les convertir en signaux compatibles TTL. Cela nous a été facilité par l'éxistence d'un circuit integré assurant cette tache de conversion. C'est le circuit MAX 238 CNG qui alimenté en 5 Volts, fournit des signaux compatibles TTL à partir de signaux compatibles RS232C et inversement. Cela nous évite l'adjonction d'une alimentation en ± 12 Volts pour la carte dans le cas d'emploie de circuits integrés assurant la même tache mais necèssitant une alimentation en ± 12 Volts (le cas des circuits MC1448 et 1489, l'un pour la conversion TTL ⟶RS232C, l'autre pour la conversion RS232C ⟶TTL). Le MAX 238 CNG est circuit integré monotension permettant de convertir des signaux compatibles TTL 0/5 Volts en signaux -10/+10 Volts compatibles RS232C.Voir schéma III.12 et annexe A.

c)Selection des vitesses d'émission et reception :

L'interfaçe serie comporte, aussi, un générateur de BAUD RATE (GBR) de type AY-5-8116, pilote par un quartz de 5.0688 MHz, permettant de fixer la vitesse d'émission et de reception avec le terminal ou le PC (cas du downloading). Le GBR delivre une frequence de reception (FR) suivant l'état des lignes RA, RB, RC et RD et une fréquence d'émision (FT)suivant l'état des pines TA, TB, TC, et TD. Le forçage des états de ces pines se fait d'une manière hard (par microswitchs). Une seconde manière de le faire serait par programme, en y ajoutant un latch ayant une adresse precise dont les sorties seront reliées aux lignes RA, RB, RC, RD et TA, TB, TC, TD. Ainsi pour choisir une vitesse donnée il suffira d'envoyer à l'adresse du latch une donnée sur huit bits correspondant à la vitesse voulue. Les lignes (RA,TA), (RB,TB), (RC,TC) et (RD,TD) ont été appairées pour avoir une fréquence d'émision egale à la fréquence de reception. N'étant encore qu'en phase de développement on a choisie de les égaliser réduisant le nombre de microswitchs et le nombre d'erreurs pouvant survenir. Le GBR délivre des fréquences allant de 110 Bauds à 19200 Bauds (bits/s).

d) Circuits d'aiguillage:

Dans le but d'effectuer des opérations de téléchargement de programmes à partir d'un micro-ordinateur,on a ajouté un circuit d'aiguillage permetant la liaison de l'émulateur EMUL186 avec uc PC. En effet, quand la demande de téléchargement est formulée par l'utilisateur ce circuit verrouille la communication avec le terminal et autorise celle avec le PC. Aucun caractère n'est envoyé au terminal ou reçu de celui-ci.
Voir schema III.12 illustrant les circuits d'aiguillage.

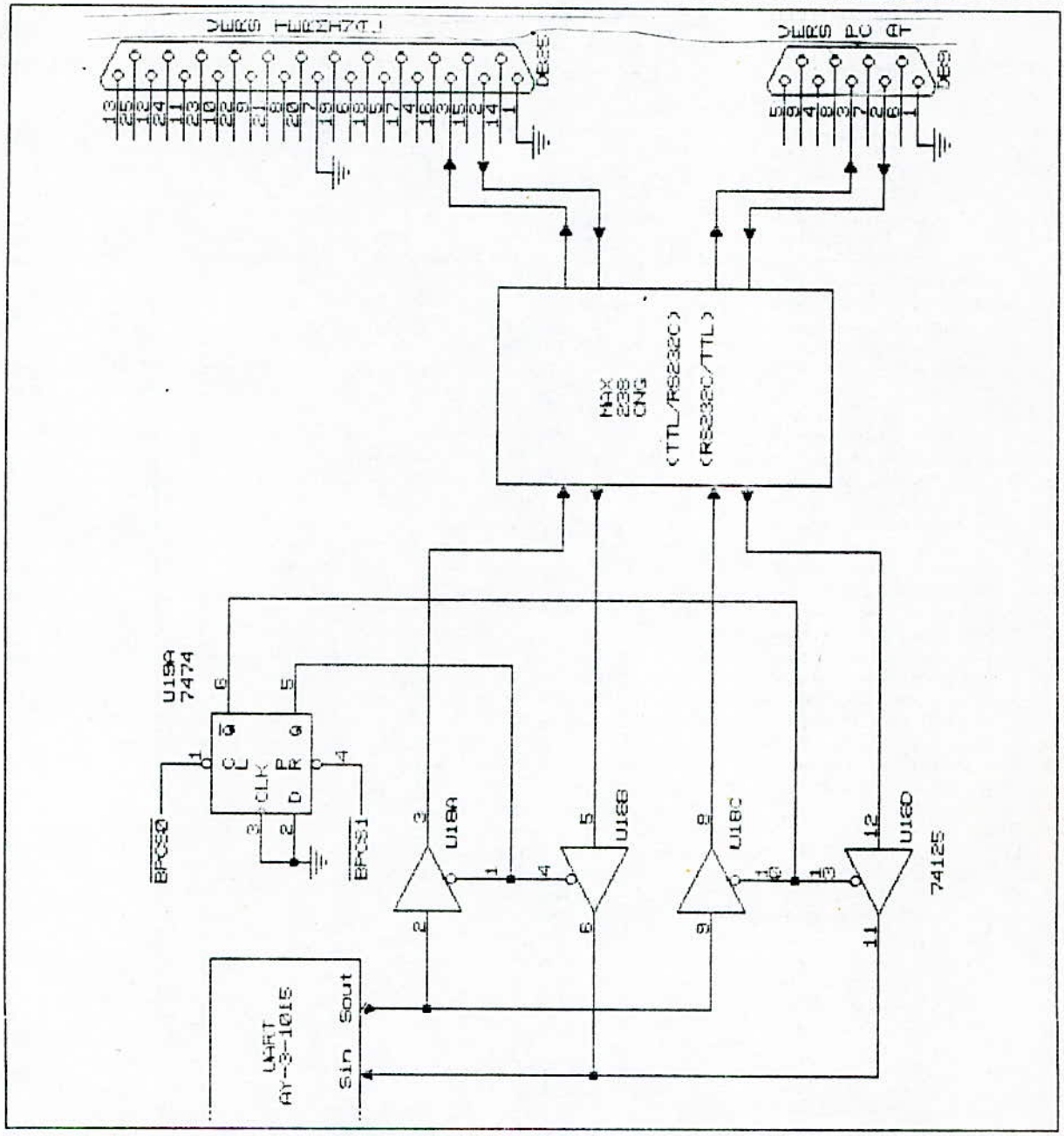


Schéma III.12 : Circuits d'aiguillage.

* Liaison EMUL186/TERMINAL.

* Liaison EMUL186/PC.

CHAPITRE IV :

PROGRAMME D'INITIALISATION.

Les microprocesseurs 8086, 8088, 80186, 80188 et 80286 contiennent les même registres de base, instructions et modes d'adressage. Néanmoins, le 80186 (et son homologue huit bits 80188) sont des microcontrôleurs integrant des périphériques. Le choix du mode de fonctionnement de ces peripheriques se fait par programmation de leurs registres. L'objectif de l'initialisation est, d'une part, la programmation de ces registres dépendant des ressources offertes par la carte (capacité mémoire,nombre d'interface...etc). D'autres part, elle concerne l'initialisation des differentes variables du moniteurs et des modes de fonctionnement.

1. PROGRAMMATION DU 80186 :

1.1. Registres d'usage general :

Le 80186 possède 14 registres d'usage général.(Voir figure IV.1).

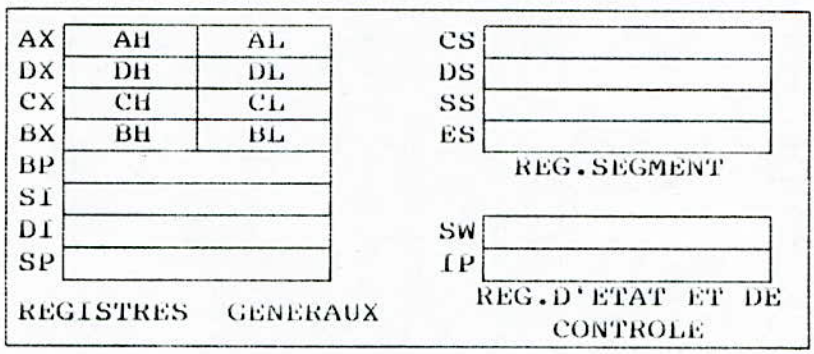


Figure IV.1 : Registres généraux du 80186.

→ Registres généraux :

Quatre registres (AX,BX,CX et DX) utilisables en registres de seize bits ou de huits bits.

→ Registres segment :

Quatre registres de segment permettent de selectionner à tout instant un bloc mémoire de 64 Kbytes .

→ Registres base et index :

Pour le déplacement à l'interieur d'un segment quatre autre registres dont deux servent d'index ont été prévus . Ces registres devront contenir des adresses de base ou d'index pour une location mémoire particulière .

→ Registre d'état et de contrôle :

Le pointeur d'instruction pointe l'instruction suivante à exécuter. Le registre d'état (Status Word register) contient les bits d'état et de contrôle dont le format est le suivant :

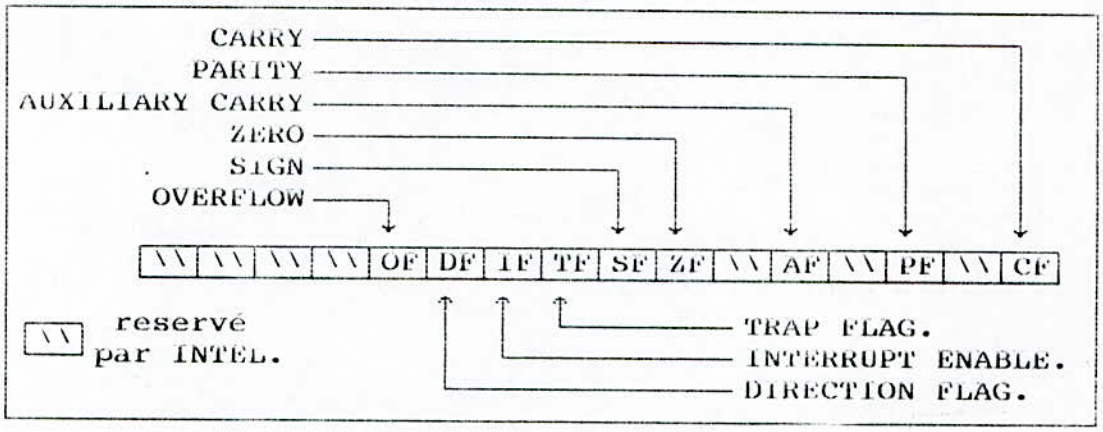


Figure IV.2 : Format de mot d'état .

1.2. Registres de contrôle :

Le 80186 contient des périphériques integrés qui sont contrôlés par le biais de registres de contrôle. Ces derniers sont contenus dans un bloc de 256 bytes appelé bloc de contrôle. Ce bloc peut etre placé dans l espace mémoire ou dans l'espace d'E/S.

→ Registre de relocation :

L'adresse de base du bloc de controle est programmable par le biais d'un registre appelé registre de relocation. Il est situé à l'offset FEH du meme bloc par rapport à l'adresse de base choisie. (Voir figure IV.3)

15	14	13	12	11	0
ET	RMX	X	M/IO	RELOCATION ADDRESS A8-A19	

ET = ESC Trap/No ESC Trap (1/0).
M/IO = register block located in Memory/ I/O Space (1/0).
RMX = Master interrupt controller mode /iRMX compatible interrupt controller mode (0/1).

Figure IV.3 : Registre de relocation.

A l'initialisation , la valeur de ce registre est 20FFH. Cette valeur permet de positionner le bloc de contrôle à l'adresse FF00H dans l'espace d'I/O. Les offsets (ou déplacements par rapport à l'adresse de base ci-dessus) des différents registres de contrôle sont données ci-dessus à la figure IV.4.

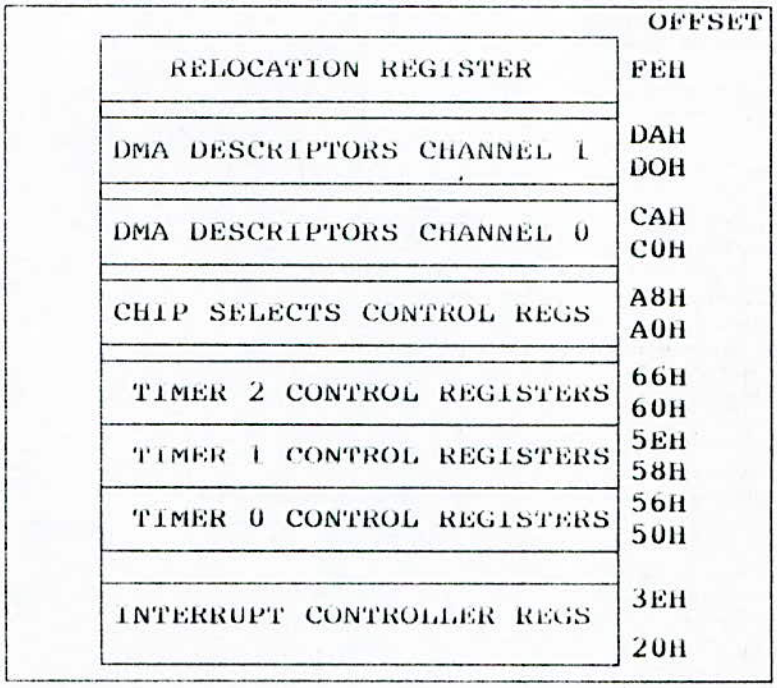


Figure IV.4 : Mapping des registres de contrôle internes.

→ Registres de contrôle de la mémoire :

Le 80186 contient une logique qui fournit des lignes de sélection programmables pour l'espace mémoire et entrées/sorties. Chaque ligne contrôle une plage d'adresses.

a) Chip select de la mémoire haute UMCS:(Upper Memory Chip Select)

La limite supérieure de la zone controlée par ce chip select est FFFFFH, cependant sa limite inférieure est programmable.
Voir figure IV.5.

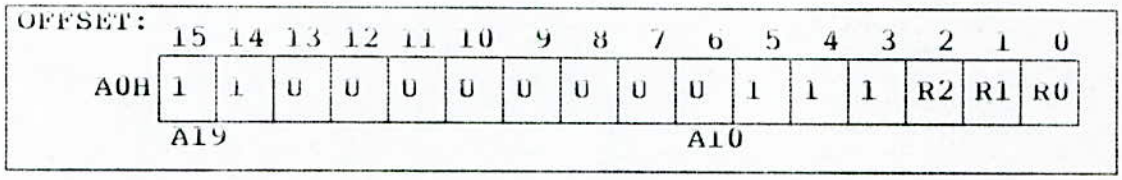


Figure IV.5 : Registre UMCS.

Ce chip-select peut valider une zone de 1,2,4,8,..,256 KBytes.

b) Chip-select de mémoire basse LMCS : (Lower Memory Chip Select)

La limite inférieure de la zone mémoire controlée par ce registre est 00000H, sa limite supérieure est programmable.

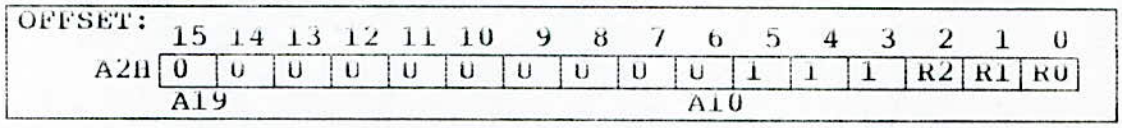


Figure IV.6 : Registre LMCS.

c) Chip-select de la memoire intermediaire MMCS,MPCS :

Le mégabytes de mémoire adressable par le 80186 est donc contrôlé, d'une part par UMCS, pour la mémoire haute, d'autre part par LMCS ,pour la mémoire basse. Cependant la zone intermediaire est controlée par deux registres (MMCS et MPCS),materialisés par quatre lignes de selection $\overline{MCS0-3}$. En effet, la partie intermediaire est découpée en quatre parties égales, controlées chacune par une ligne de selection La dimension de ces parties est programmable par le registre MPCS.

Voir figure IV.7.

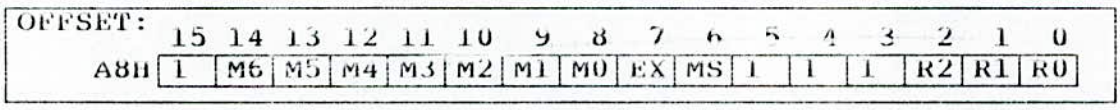


Figure IV.7 : Registre MPCS.

Les bits 8 à 14 (M0 à M6 respectivement) permettent de choisir la dimension des parties constituant la mémoire intermédiaire. Un bit et un seul est mis à "1" à la fois. L'adresse de base de cet espace est définie au moyen du registre MMCS.Voir figure IV.8.

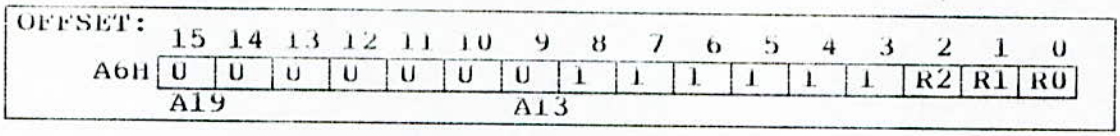


Figure IV.8 : Registre MMCS.

→ Registres de contrôle des périphériques PACS et MPCS :

Le 80186 génére sept lignes de selection peripheriques. Ces chip- selects sont actifs pour sept blocs continus de 128 bytes a partir de l'adresse de base programmée. Ces blocs peuvent etre situés soit dans l'espace entrées/sorties soit dans l'espace mémoire. L'adresse de base (PBA : Programmable Base Address) ne peut etre qu'un multiple de UN Kilobytes. L'adresse de base de ces blocs est définie par le registre PACS. Voir figure IV.9.

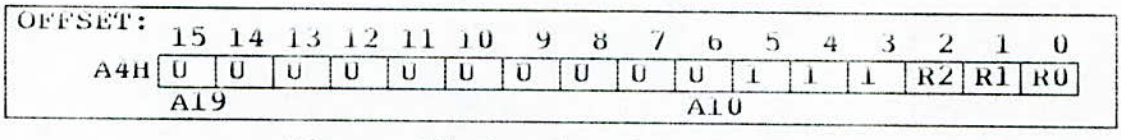


Figure IV.9 : Regsitre PACS.

Le mode d'opération des PCS est définie par le registre MPCS (ce même registre définie, aussi, la dimension des parties de la mémoire intermédiaire). Le bit 6 permet de situer le bloc des périphériques dans l'espace d'E/S ou dans l'espace memoire.

→ Logique de génération d'états d'attente:(READY generation logic)

Les bits R0,R1 & R2 des registres UMCS, LMCS, MPCS, MMCS et PACS permettent d'une part de définir le nombre d'états d'attente à inserer dans le cycle bus pour chaque chip-select(ce nombre varie de 0 à 3 suivant les valeurs de R0 et R1) ,d'autre part de valider ou d'interdire un signal d'etat d'attente externe .

→ Registres de contrôle des canaux DMA :

Chaque canal DMA est munis de six registres de contrôle situes dans le bloc des registres de contrôle.Ces registres fixent les modalités de fonctionnement de chaque canal .Chaque canal DMA est controlé par :

- un pointeur source de 20 bits (deux mots de 16 bits);

- un pointeur destination de 20 bits (deux mots de 16 bits);
- un compteur de transfert à 16 bits;
- et le mot de contrôle de 16 bits .

Le compteur de transferts (TC) specifie le nombre de transferts à effectuer (TC peut atteindre 64 mille transferts d'octets ou de mots). Le mot de contrôle définie le canal opérationnel.

→ Registres de contrôle des TIMERS :

Le 80186 possède trois timers integrés programmables à 16 bits . Deux d'entre-eux T0,T1 possedent chacun une entrée ,qui leur permet d'etre utilisés en compteur d'evenements externes. Ils peuvent ,aussi, etre utilisés pour mesurer la durée d'un evenement externe et générer une forme d'onde non répétitive. Le troisième timer est utilisé pour les applications temps réel et en tant que monostable. De plus, le troisième timer peut fonctionner en echelonneur ("PRESCALER") pour les deux autres timers ou pour declencher une demande de DMA Le registre compteur contient la valeur courante du timer Cette valeur est incrementee pour chaque evenement. Chaque timer contient un registre de valeur maximale (Max count register) qui une fois atteinte le registre compteur est remis à zéro.

→ Registres de contrôle du contrôleur d'interruption :

Le 80186 possede un certain nombre d'entrées d'interruption (certaines internes, d'autres externes). Le contrôleur d'interruption integré est, en fait, un 8259A classique. Cinq lignes d'interruption sont fournies à l'extérieur , dont une non masquable (NMI).

Le contrôleur d'interruption sera détaile dans ce chapitre.

MODES D'OPERATION DU CONTROLEUR D'INTERRUPTION :

a) Fully nested mode ou mode direct :

Dans ce mode quatre lignes sont utilisées directement comme demande d'interruption .

b) Mode en cascade :

Des quatre lignes d'interruption deux sont utilisées comme paire demande/ prise en compte de la demande d'int. La pine INT0 est

branchée à un contrôleur d'interruption 8259A externe. La pine INT2/$\overline{\text{INTA0}}$ sert de ligne de prise en compte (ou d'accusé de reception) de l'interruption envoyée par le 8259A externe.Idem pour INT1 et INT3/$\overline{\text{INTA1}}$.

c) Mode "Special fully nested mode" :

Ce mode permet l'emboitement de plusieurs contrôleurs d'interruption externes.

REGISTRES DE CONTROLE :

Le contrôleur d'interruption contient quinze registres de contrôle.Voir ci-dessous l'ensemble des registres de contrôles du controleur d'interruption :

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA1 CONTROL REGISTER	36H
DMA0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REG.	30H
INTERRUPT REQUEST REG.	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure IV.10 : Registres du contrôleur d'interruption.

a) Registre In-Service :

Ce registre contient un bit de service "In-Service Bit" pour chaque source d'interruption.Voir figure IV.11.

15	14	...	10	9	8	7	6	5	4	3	2	1	0
0	0	...	0	0	0	I3	I2	I1	I0	DI	D0	0	TMR

Figure IV.11 : Format des registres en-service,demande d'interruption et du registre masque.

Le bit TMR est le bit de service des timers,D0 et D1 pour les 2 canaux DMA (DMA 0 et DMA 1 resp.).Les bits 10-13 sont ceux des quatre interruptions masquables externes (INT0-3 resp.). b) Registre de demande d'interruption : Le format de ce registre est

identique à celui du précédent. Ce registre indique l'état des lignes d'interruption timer, DMA et interruptions externes.

c) Registre de masque :

Ce registre contient un bit masque pour chaque source d'interruption.Son format est identique à celui du registre en service et de demande d'interruption.

d) Registre masque de priorité :

Ce registre masque toutes les interruptions en dessous d'un certain niveau de priorité .Le format de ce registre est montré sur la figure IV.12:

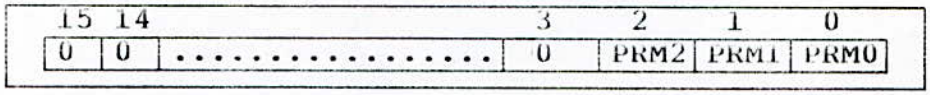


Figure IV.12 : format de registre masque de priorite.

Les bits PRM 0,1 et 2 forment le niveau de priorité.

e) Registre d'état des interruptions :

Ce registre contient l'état des interruptions.Voir figure IV.13.

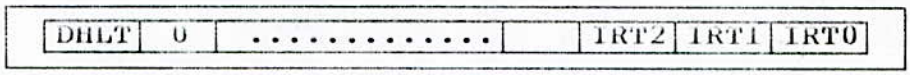


Figure IV.13 : Registre d'état d'interruption.

Le bit DHLT : DMA Halt transfer .Ce bit est mis à "1" lors des transferts DMA. Les bits IRT0,1,2: indiquent les demandes d'interruption des timers.

f) Registres de contrôle timer ,DMA 1 ET 0 :

Ces registres forment les mots de contrôles pour toutes les interruptions internes. Voir figure IV.14.

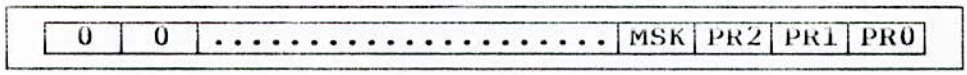


Figure IV.14 : registre de contrôle timer/dma.

PR 2,1 et 0 : représentent le niveau de priorité.

MSK : bit d'inhibition des demandes d'interruptions

g) Registres de contrôle des interruptions INT0-3 :

Les valeurs de ces registres constituent les mots de commandes pour chaque interruption

Voir figures IV.15 et IV.16.

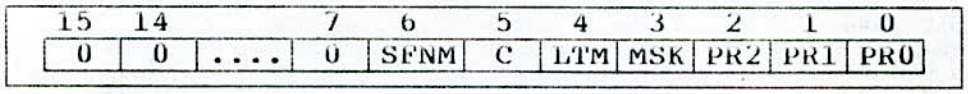


Figure IV.15 : Registre de contrôle int0/int1.

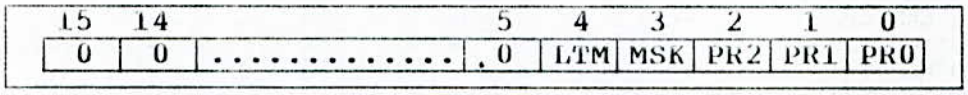


Figure IV.16 : Registre de contrôle int2/int3.

PR0-2 : niveau de priorité = 111 → minimale;
= 000 → maximale.

LTM : bit du mode de déclenchement de l'interruption,
LTM = 1 → declenchement par niveau;
= 0 → declenchement par front montant.

MSK : bit de masque,
MSK = 1 → "validation" de masque;
= 0 → pas de masque .

C : choix du mode du contrôleur d'interruption.
C=1 → cascade mode (mode en cascade);
C=0 → fully nested mode (mode direct).

SFNM : bit de validation du mode "Special Fully Nested Mode".
SFNM = 1 mode validé.

h) Registre fin d'interruption EOI :(EOI : End OF Interrupt)

Il permet de déclencher une commande de fin d'interruption en écrivant (dans ce registre) une valeur adéquate
Voir figure IV.17.

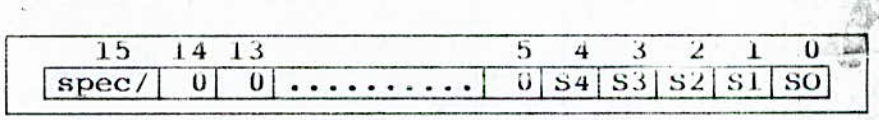


Figure IV.17 : registre fin d'interruption.

S0-4 : bits constituants le type de l'interruption ;

SPEC/ : ce bit determine le type de la commande de fin d'interruption ,
SPEC/ = 1 → mode non spécifique ;
= 0 → mode spécifique (dans ce cas, il faudra spécifier le type de l'interruption).

i) Registres d'états de polling et de polling :(polling = scrutation)

La lecture du registre de polling constitue le polling lui-même. Cette lecture positionne a "1" le bit de service de l'interruption la plus prioritaire. Voir figure IV.18.

15	14	13		5	4	3	2	1	0
intreq	0	0	.....	0	S4	S3	S2	S1	S0

Figure IV.18 : Registre de polling.

S0-4 : indiquent le type de l'interruption la plus prioritaire;

INTREQ : Ce bit indique si une demande d'interruption a été formulée ,

INTREQ = 1 → demande interruption présente;

INTREQ = 0 → pas de demande d'interruption.

2. INITIALISATION :

La routine d'initialisation consiste à programmer les differents blocs internes ou externes au microcontrôleur.

Actuellement, la carte comprends en grosso modo :

- un module mémoire ROM de 8 KBytes (2 EPROMs de 4 KBytes chacune) necessaire pour contenir la routine d'initialisation, le moniteur de gestion et le debugger.
- un module memoire de RAMs statiques de 64 KBytes (soit 2 RAMs de 32 KBytes chacune) dedies pour contenir les variables du moniteur et du debugger ,ainsi, que les programmes à tester.
- un module de communication servant d'interface entre la carte et un terminal ou un PC.

A la mise sous tension la première instruction à exécuter se trouve a l'adresse FFFF0H, avec le contexte suivant:

REGISTERS	VALUES
STATUS WORD	F002H
INSTRUCTION POINTER	0000H
CODE SEGMENT	FFFFH
DATA SEGMENT	0000H
EXTRA SEGMENT	0000H
STACK SEGMENT	0000H
RELOCATION REGISTER	20FFH
UMCS REGISTER	FFFBH

Figure IV.19 : Contexte du 80186 aprés le RESET.

A l'initialisation seul l'espace du dernier KBytes, du MégaBytes adressable, est validé. Toutes les initialisations, concernant l'espace mémoire et l'espace d'entrée/sortie, devront etre faites avant de quitter cet espace, car tout accès à un espace mémoire non validé entrainera le plantage du microcontrôleur.

2.1. Initialisation du registre de relocation :

A l'initialisation le registre de relocation a la valeur 20FFH. Cette valeur correspond au placement du bloc des registres de contrôle dans l'espace d'E/S à l'adresse FF00H. Cette valeur du registre de relocation est celle qui est maintenue dans notre initialisation. En effet cette valeur semble etre sans aucune gêne pour l'étape actuelle du développement. Cette valeur configure le controleur d'interruption integre en mode maitre.

2.2. Initialisation des registres de contrôle mémoire:

La carte contient deux EPROMs situées dans la partie haute de la mémoire et deux RAMs situées dans la partie basse de celle-ci. Le registre correspondant à la partie haute de la mémoire est UMCS celui-ci est mis à la valeur FE38H. La limite superieure de cette zone est FFFFFH, la limite inférieure est FE000H. Les EPROMs utilisées sont suffisemment rapides elles ne necéssitent, donc, pas d'états d'attente. Idem pour les RAMs , le registre de contrôle de celles-ci est LMCS. Ce dernier est initialisé à la valeur 0FF8H validant les 64 KBytes sans insertion d'états d'attente. La limite inférieure est 0 et la limite supérieure est 0FFFFH. La mémoire intermédiaire n'a pas été encore installée mais néanmoins on le programme, comme même, pour une éventuelle addition de mémoire

sans modifier la présente routine d'initialisation. Le registre MMCS a été initialisé à la valeur 09F8H sans états d'attente validant seize KBytes. Le registre MPCS validant la dimension des blocs de la mémoire intermédiaire est mis à 8238H.

2.3. Initialisation des registres de contrôle des périphériques:

Les registres de contrôle des périphériques MPCS et PACS sont initialisés à 8238H et 0038H respectivement. Les périphériques sont placés dans l'espace d'E/S avec deux états d'attente et prise en compte du signal du READY externe.

2.4. Initialisation des registres de contrôle DMA:

La validation du mode DMA se fait par mise à un du bit START/STOP dans le registre de contrôle. La demande de DMA peut etre générée de façon périodique (cas du rafraichissement des mémoires dynamiques). Avant de valider le mode DMA les pointeurs source et destination et le registre compteur de transferts devront etre programmés aux valeurs voulues. Lors de l'initialisation les registres de contrôle DMA sont mis à zéro de sorte à ne pas générer d'interruption lors de cette étape.

2.5. Initialisation des registres de contrôle des timers:

Idem pour les registres de contrôle des timers qui sont programmées de façon à ne pas generer d'interruption lors de la routine d'initialisation.

2.6. Initialisation du registres du controleur d'interruption:

Le contrôleur d'interruption possede trois mode de fonctionnement :

- Fully Nested Mode (mode direct);
- Cascade Mode (association de contrôleur d'interruption externe);
- Special Fully Nested Mode (association de plusieurs contrôleurs d'interruption).

A l'étape actuelle de développement de la carte, le nombre d'interruptions externes étant. De ce fait, on a opté pour le "fully

nested mode" ou le mode direct. Dans ce mode la ligne d'interruption est directement reliée à la source d'interruption.
Remarque: Dans une phase plus avancée du développement de cet émulateur on y ajoutera un ou deux contrôleurs d'interruption pour avoir au moins huits lignes d'interruption, d'une part pour envisager la compatibilité avec IBM, d'autre part, pour donner à l'utilisateur de cet émulateur toutes les ressources que lui offre ce microcontrôleur.

La ligne d'entrée d'interruption utilisée est INT0. Pour cela, les bits à positionner sont :

- le bit C de registre INT0 est mis à zéro validant le mode direct;
- le bit SFNM est mis à zéro dévalidant le mode special fully nested mode;
- le bit LTM est aussi mis à zéro choisissant le déclenchement de l'interruption sur front montant du signal externe d'interruption.
- le bit Spec/ du registre fin d'interruption est mis à un pour valider le mode non spécifique afin d'indiquer la fin d'interruption. Dans le mode non spécifique on ne spécifie pas le type d'interruption qui a été exécutée.

Le programme d'initialisation est accompagné d'un petit programme de lecture/écriture. Ce programme consiste en la lecture du port d'E/S relié au terminal qui correspond à la touche appuyée, puis son renvoi vers l'écran. Il ne reste qu'à le charger en EPROMs. Le programme a été saisie sur PC puis assemblé et linké.

Le programme exécutable est ensuite mis sous format Intel-Hexa pour le transmettre vers le programmateur d'EPROMs via une liaison série. Cette mise en forme se fait grâce à un programme fait en turbopascal, consistant à n'extraire que le programme proprement dit situés à l'offset 200H par rapport au début du fichier exécutable. En d'autres termes, il élimine les informations concernant la taille du fichier et les autres.

En annexe F sont illustrées les différentes methodes de chargement de programmes dans l'émulateur.

CHAPITRE V :

UN MONITEUR DE GESTION "MON186".

Le moniteur est l'ancienne appélation de système d'exploitation. Le mot moniteur est surtout utilisé quand il s'agit d'un systeme à microprocesseur relativement simple. Les systemes d'exploitation servent d'intermediaire entre le materiel et son utilisateur. En effet,le système d'exploitation gère les différentes ressources d'un système à microprocesseur au sein duquel il a été instalé.

1. Généralités sur les systèmes d'exploitation:

Deux fonctions essentielles dans système d'exploitation:

- L'initialisation et la gestion des echanges "processeur-périphérie"

Cette fonction comprends l'initialisation du processeur et des canaux d'E/S, la gestion des interruptions et des communications.

Cette fonction, appelée noyau, est directement liée au hardware. (elle a été l'objet du chapitre précedent).

- La mise à la disposition de l'utilisateur de l'ensemble des ressources software et hardware. Cette partie offre à l'utilisateur la possibilité d'ecrire, modifier, exécuter et stocquer des programmes et des donnees.

On distingue cinq sous-fonctions importantes (Voir Figure V.1):

* l'interface utilisateur/machine qui gère les périphériques de dialogue;
* stockage permanent de l'information ;
* gestion de l'information ;
* exécution des programmes.

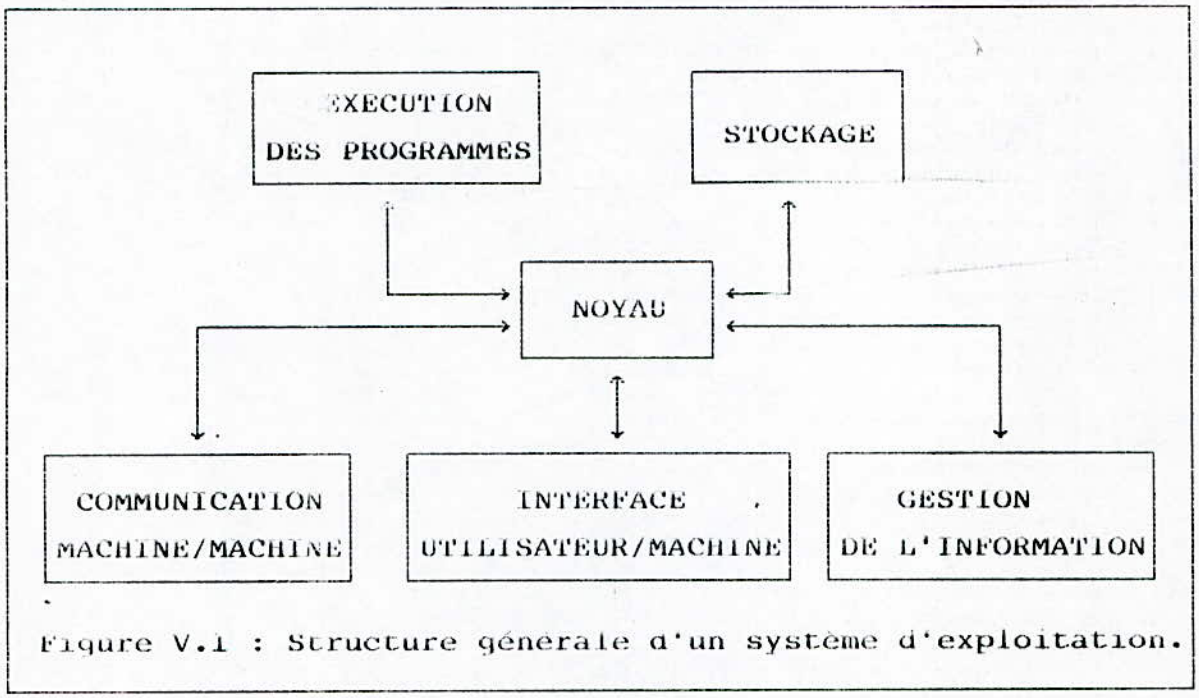


Figure V.1 : Structure générale d'un système d'exploitation.

2. STRUTURE DE MON186:

Le moniteur MON186 est formé de routines hierarchisées en trois niveaux (Voir figure V.2):

- Premier niveau:

Ce niveau consiste en une boucle infinie d'attente,de reconnaissance et d'exécution de commandes.

- Deuxième niveau:

Il comprends toutes les routines, exécutables, accéssibles à l'utilisateur.

- Troisième niveau:

Il est constitué par l'ensemble des routines élementaires transparentes à l'utilisateur. Ces routines sont utilisées par les différentes commandes.

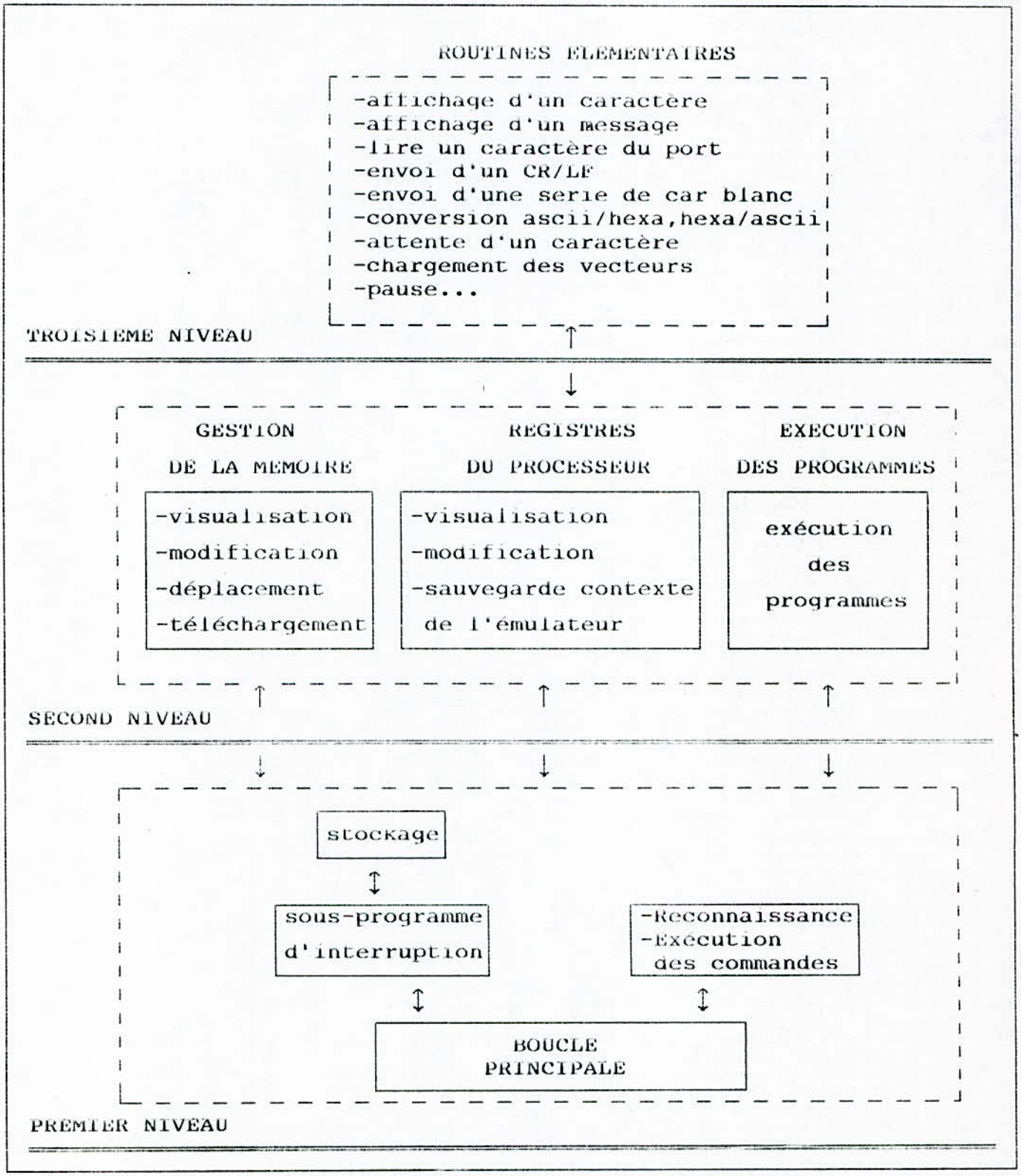


Figure V.2 : La structure de MON186.

3. Les commandes de MON186:

Il existe deux sortes de commandes

- → Commandes à paramètres;
- → Commandes sans paramètres.

Format d'une commande:

<Nom de la commande> parametre1 parametre2 parametre3

Exemple: MOVE 0000:0000 0000:0100 0000:1000

	adresse début	adresse fin	adresse destination

A chaque commande correspond un bit dans un mot de 16 bits appelé STATUS_CMD informant, en permanance, le moniteur sur de la commande en cours d'execution. Au début de l'execution de chaque commande le bit correspondant est mis à 1. Il est remis à zéro à la fin de celle-ci. Certaines commandes restent à l'écoute du clavier pour decider de leur continuité ou de leur suspension.
Exemple: la commande DUMPALL, qui permet de visualiser toute la mémoire par page de 100H.

La touche '+' permet d'afficher la page suivante

'-' la page précendente.

Les touches 'S' et 's' permettent de sortir de cette commande.

La gestion des commandes (et le téléchargement de programmes) necessite la gestion d'une FIFO (First In First Out).

3.1. Routines du premier niveau:

Elles constituent le corps du programme autour duquel satellitent toutes les routines de MON186. Ce niveau est, en fait, une boucle infinie d'attente d'une commande, puis de sa reconnaissance et ensuite de son execution. L'organigramme correspondant est donné à la figure V.1.

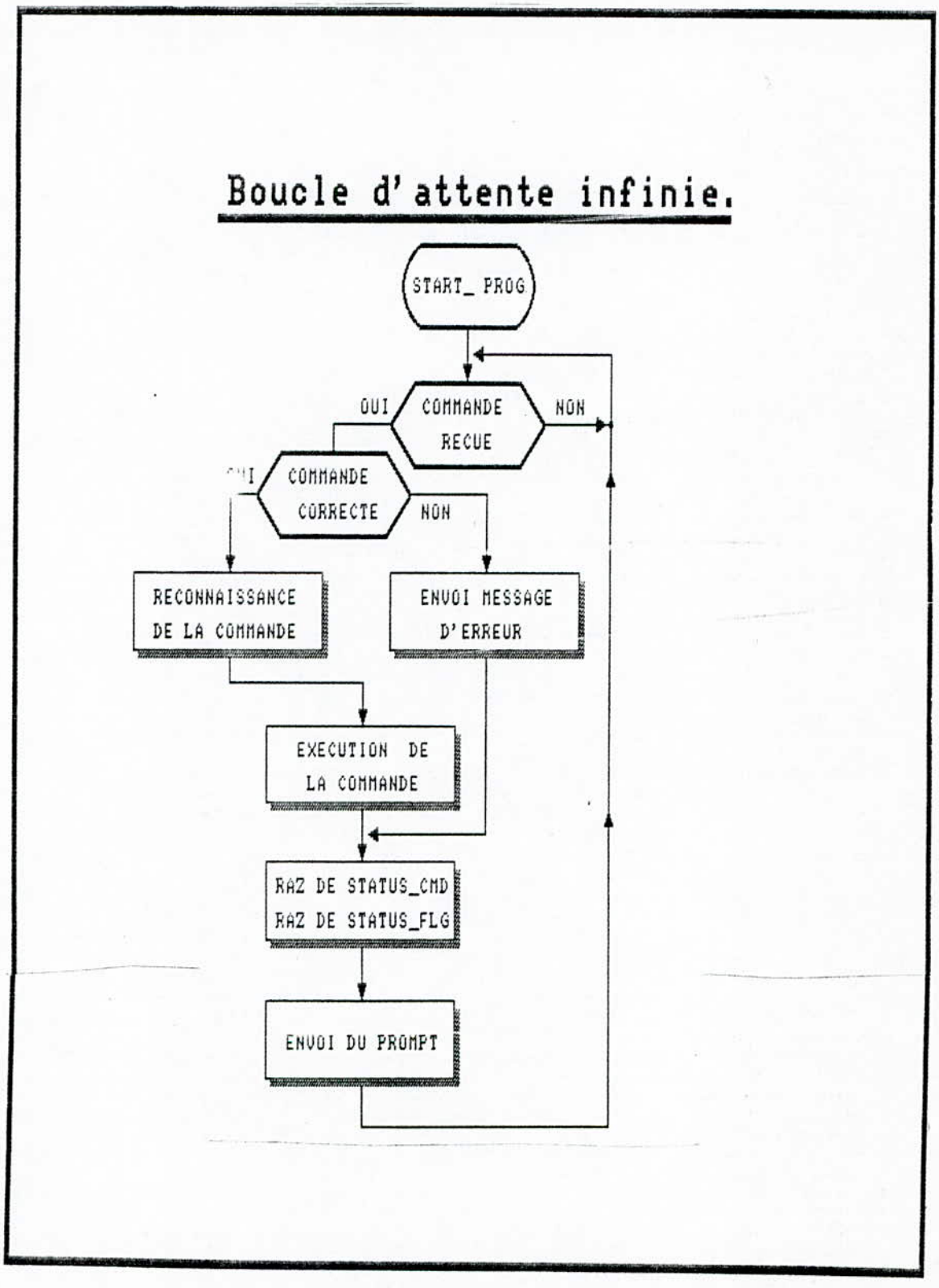


Figure V.1 : Organigramme de la boucle principale.

Cette boucle principale travaille avec trois routines qui sont :

- le traitant de l'interruption clavier;
- la routine de reconnaissance et d'exécution des commandes;
- et la routine de gestion de la FIFO.

3.1.1. Traitant de interruption clavier:

Les commandes sont entrées par le biais du clavier, lié à l'unité centrale par la ligne d'interruption INT0. Le caractère lu est ensuite traité. La figure V.2 montre l'organigramme du traitant de l'interruption clavier.

3.1.2. Routine de stockage :

La FIFO est le buffer du port d'entrée (lié au clavier et au micro-ordinateur). Toutes le informations reçues sont, temporairement, stocquées. L'organigramme de gestion de la FIFO est donné à la figure V.3.

La FIFO est acceder au moyen de deux pointeurs :

-un pointeur d'ecriture appelé PTR WRITE;

-un pointeur de lecture appelé PTR_READ.

Déclaration de la FIFO :

```
START_OF_FIFO   EQU  THIS BYTE
CMD_BUFFER      DB   256 DUP (?)
END_OF_FIFO     EQU THIS BYTE
```

3.1.3. Routine de reconnaissance des commandes :

Cette routine, sert à verifier l'existence de la commande reçue dans une table dans laquelle sont repertoriés tous les noms de commandes. Si cette commande existe, le programme lance son exécution aprés avoir chargé l'adresse correspondante.

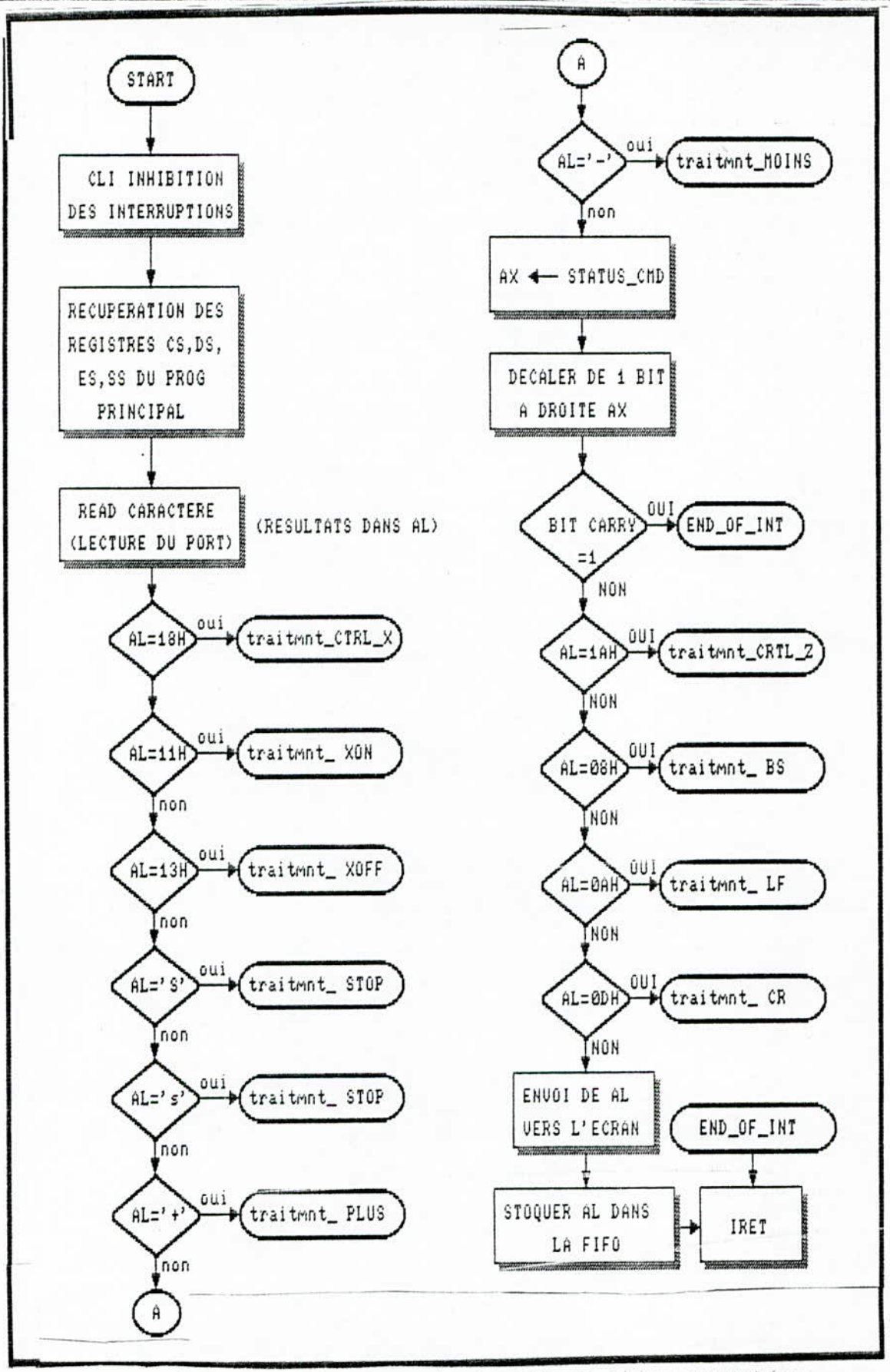


Figure V.2 : Organigramme du traitant d'interruption.

Interruption clavier.

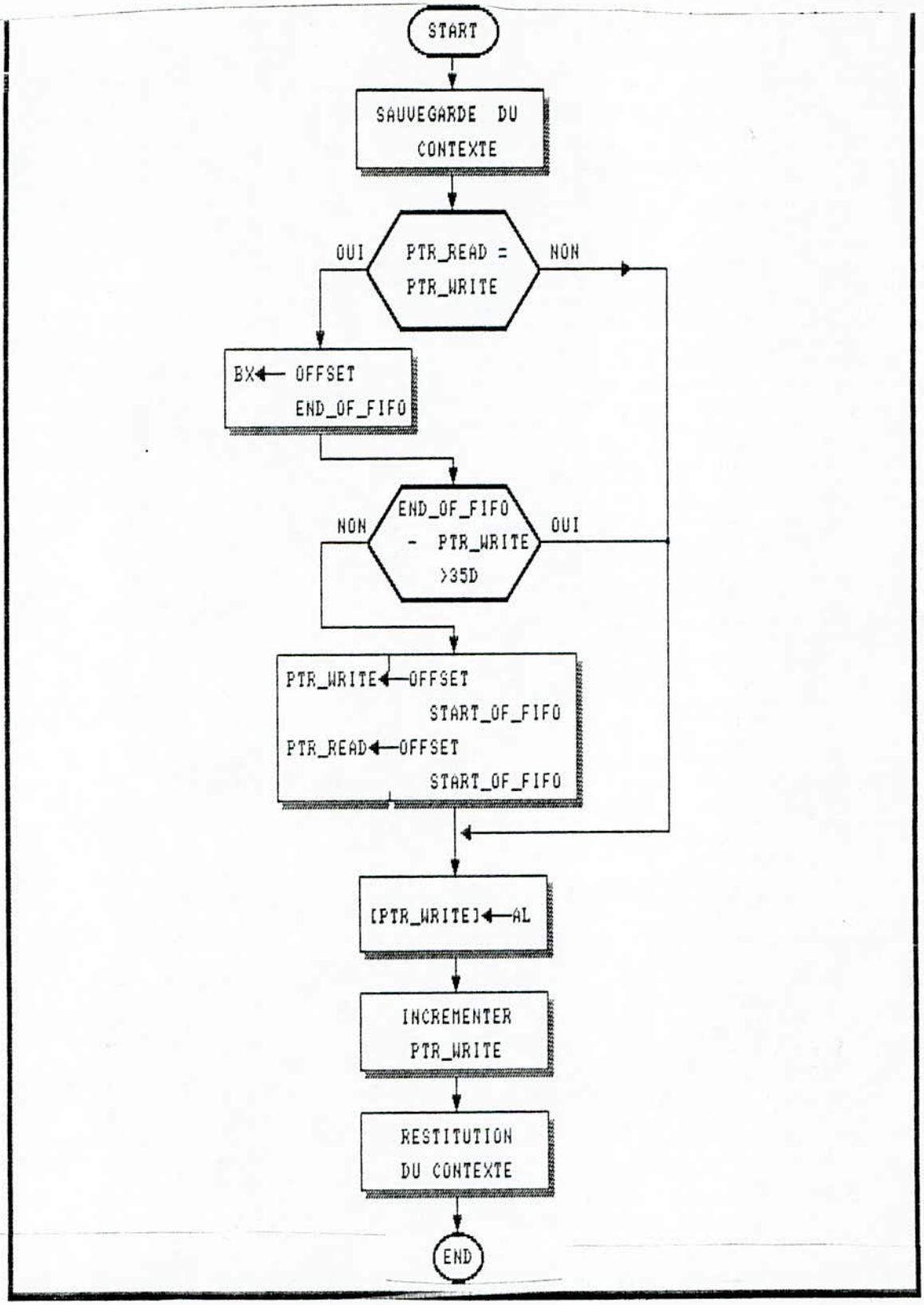


Figure V.3 : Organigramme de la routine de gestion de la FIFO et de stockage.

Procedure STORAGE.

Declaration de la table des commandes et la table des adresses:

```
COMMAND_TABLE   DB      'SREG'
                DB      'sreg'
                DB      'CREG'
                DB      'greg'
                DB      'AREG'
                DB      'areg'
                 :       :
                DB      'DUMP'
                DB      'dump'
                DB      'DLOD'
                DB      'dlod'
                DB      'INIT'
                DB      'init'

ADR COMMAND_TABLE DW    DOWNLOAD
                DW      DOWNLOAD
                DW      INIT
                DW      INIT
                DW      DUMPALL
                DW      DUMPALL
                 :       :
                DW      ALL REGISTERS
                DW      ALL REGISTERS
                DW      GENERAL REGISTERS
                DW      GENERAL REGISTERS
                DW      SPECIAL REGISTERS
                DW      SPECIAL REGISTERS
```

On constatera que dans la deuxieme table l'ordre des commandes est inversé. Cela est dû à une astuce dans le programme de reconnaisance pour retrouver l'adresse de la commande reçue.

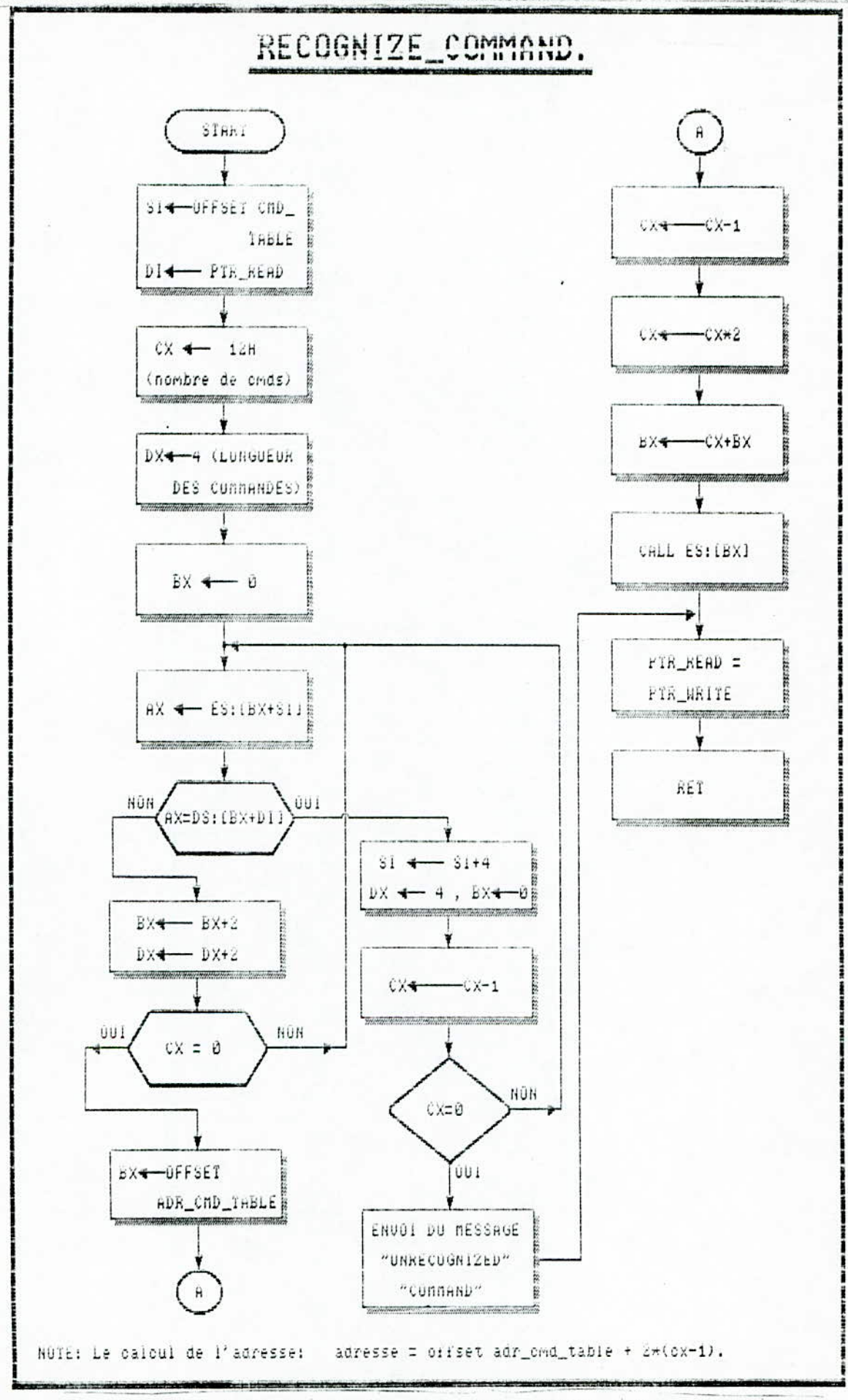


Figure V.4 : Routine de reconnaissance des commandes.

3.2. Routines du niveau 2 de MON186:

Ce sont les routines des différentes commandes accéssibles à l'utilisateur.

Elles se subdivisent en trois categories :

- Opérations sur la mémoire : modification, visualisation, téléchargement, déplacement...etc.
- Opérations sur les registres du microcontrôleur: modification et visualisation...etc
- Opérations sur les programmes de test: exécution...etc

3.2.1. Operations sur la memoire :

Les routines realisees concernant la mémoire sont:

DUMPALL : permet de visualiser toute la mémoire par page de 100H

DOWNLOAD: celle-ci permet de télécharger des programmes à partir d'un micro-ordinateur.

3.2.2. Opération sur les registres :

Les commandes ALL_REGISTERS, GENERAL REGISTERS et SPECIAL REGISTERS permetent de de visualiser le contenu des registres.

ALL_REGISTERS : affiche tous les registres;

GENERAL_REGISTERS : affiche les registres généraux et offre la possibilité de modifier un registre;

SPECIAL_REGISTRES : Affiche les valeurs des registres de contrôle) et offre la possibilité de les modifier.

3.2.3. Opération sur les programmes utilisateurs:

La commande RUN PROGRAMME (RUNP) a été prévue à cet effet. Le programme téléchargé est mis en RAM, le contenue du début de la RAM (contenant les variables de MON186 et les vecteurs des interruptions) est sauvegarde dans une zone interdite à l'utilisateur.

Deux options pour cette commande :

Execution: - en pas à pas;

- avec points d'arrêt.

Ces commandes ne sont pas encore implementées sur MON186.

3.3.Routines du troisieme niveau :

Dans ce niveau se regroupent toutes les routines élementaires utilisees par les deux niveaux précedents.

Entre autres les routines suivantes :

-affichage d'un caractère;

-affichage d'un message;

-lire un caractère du port;

-envoi d'un CR/LF;

-envoi d'une serie de car blanc;

-conversion ascii/hexa,hexa/ascii;

-attente d'un caractère;

-chargement des vecteurs;

-pause...etc

Voir organigr mmes en annexe.

CONCLUSION GENERALE

Au terme de cette étude, qui m'a été trés profitable. J'ai eu l'occasion d'approfondir mes connaissances dans le domaine des micro-ordinateurs et des systèmes à microprocesseur en général.

Je ne prétends pas avoir réalisé un émulateur complet (il reste à rajouter le nécèssaire pour la sonde d'émulation). Mais, néanmoins, la carte EMUL186 et le moniteur MON186 développés y constituent l'ébauche principale.

J'éspere que mon travail aura une suite pour aboutir à l'émulateur final cela pourra rentrer das le cadre d'un autre projet. On pourra, aussi, l'étendre à faire de l'émulation assisté par micro-ordinateur.

ANNEXES

→ MICROCONTROLEUR 80186.

* BROCHAGE.

intel®

80186
HIGH INTEGRATION 16-BIT MICROPROCESSOR

- Integrated Feature Set
  - — Enhanced 8086-2 CPU
  - — Clock Generator
  - — 2 Independent DMA Channels
  - — Programmable Interrupt Controller
  - — 3 Programmable 16-bit Timers
  - — Programmable Memory and Peripheral Chip-Select Logic
  - — Programmable Wait State Generator
  - — Local Bus Controller
- Available in 10 MHz (80186-10) and 8 MHz (80186) Versions
- High-Performance Processor
  - — 4 MByte/Sec Bus Bandwidth Interface @ 8 MHz
  - — 5 MByte/Sec Bus Bandwidth Interface @ 10 MHz
- Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O
- Completely Object Code Compatible with All Existing 8086, 8088 Software
  - — 10 New Instruction Types
- Complete System Development Support
  - — Development Software: ASM 86 Assembler, PL/M-86, Pascal-86, Fortran-86, C-86, and System Utilities
  - — In-Circuit-Emulator (I^2ICE™-186)
- Numerics Coprocessing Capability Through 8087 Interface
- Available in 68 Pin:
  - — Plastic Leaded Chip Carrier (PLCC)
  - — Ceramic Pin Grid Array (PGA)
  - — Ceramic Leadless Chip Carrier (LCC)

  (See Packaging Outlines and Dimensions, Order # 231369)
- Available in EXPRESS
  - — Standard Temperature with Burn-In
  - — Extended Temperature Range (−40°C to +85°C)

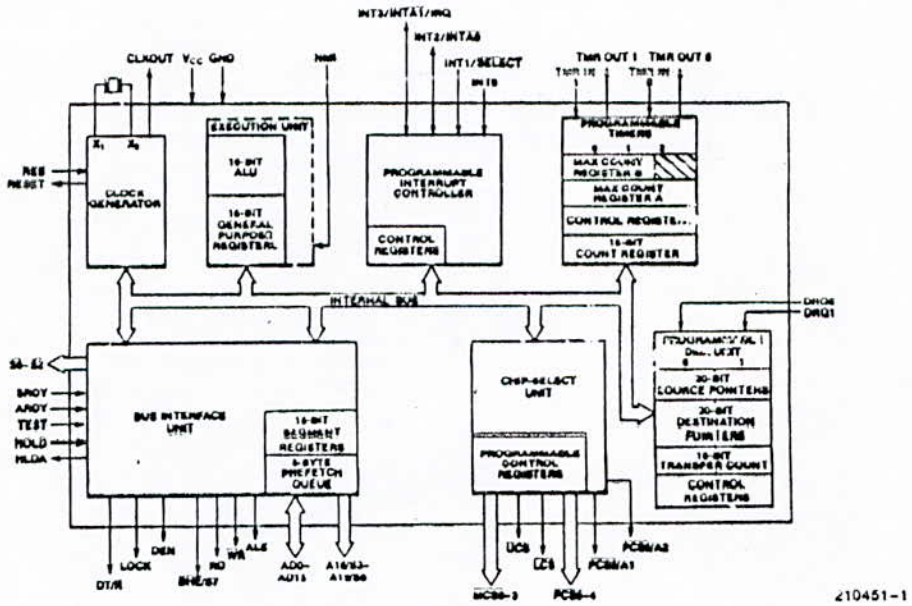


Figure 1. 80186 Block Diagram

Order Number: 210451-011

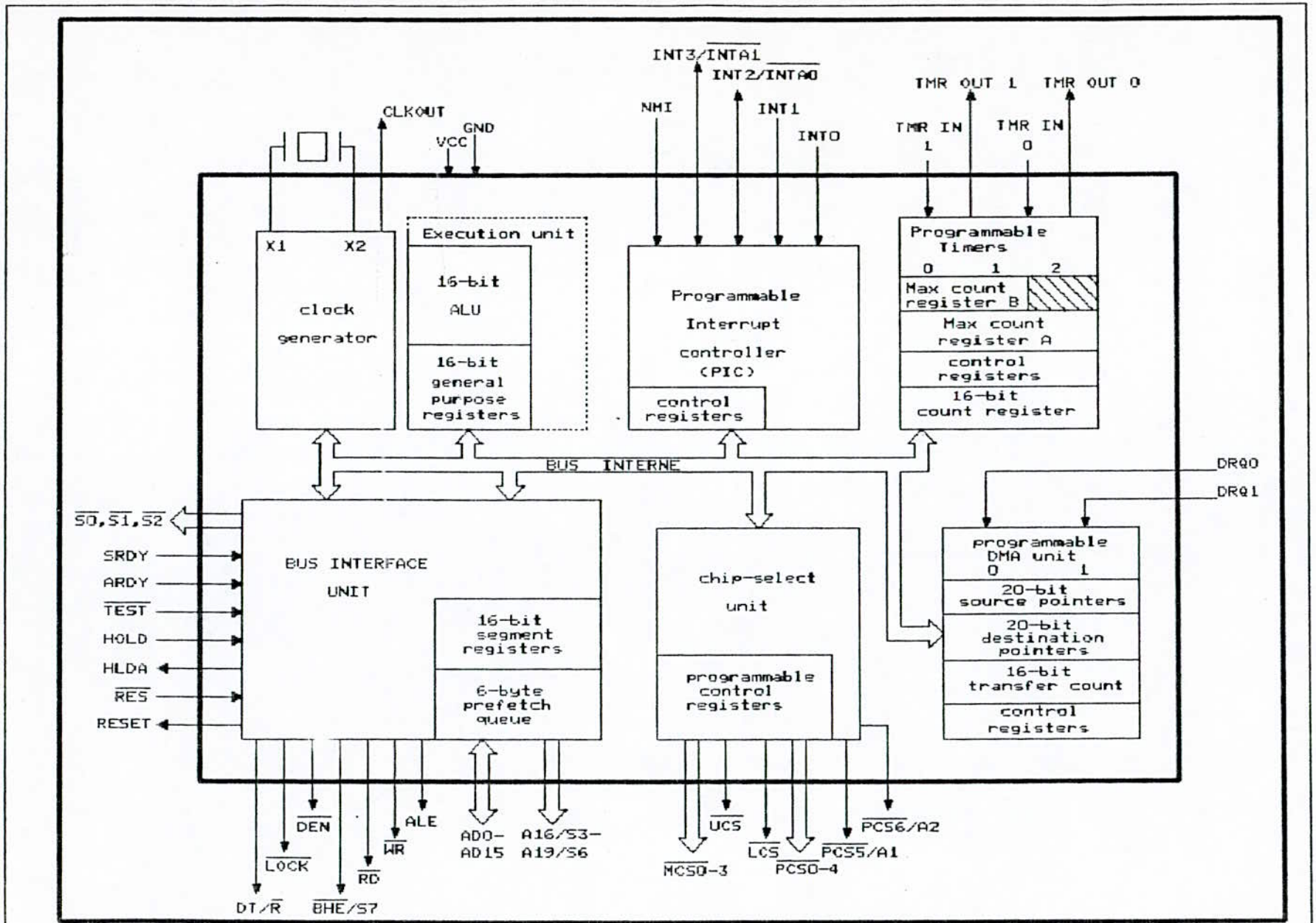


FIGURE III.1 : BLOC DIAGRAMME DU i80186.

The Intel 80186 is a highly integrated 16-bit microprocessor. The 80186 effectively combines 15–20 of the most common 8086 system components onto one. The 80186 provides two times greater throughput than the standard 5 MHz 8086. The 80186 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.

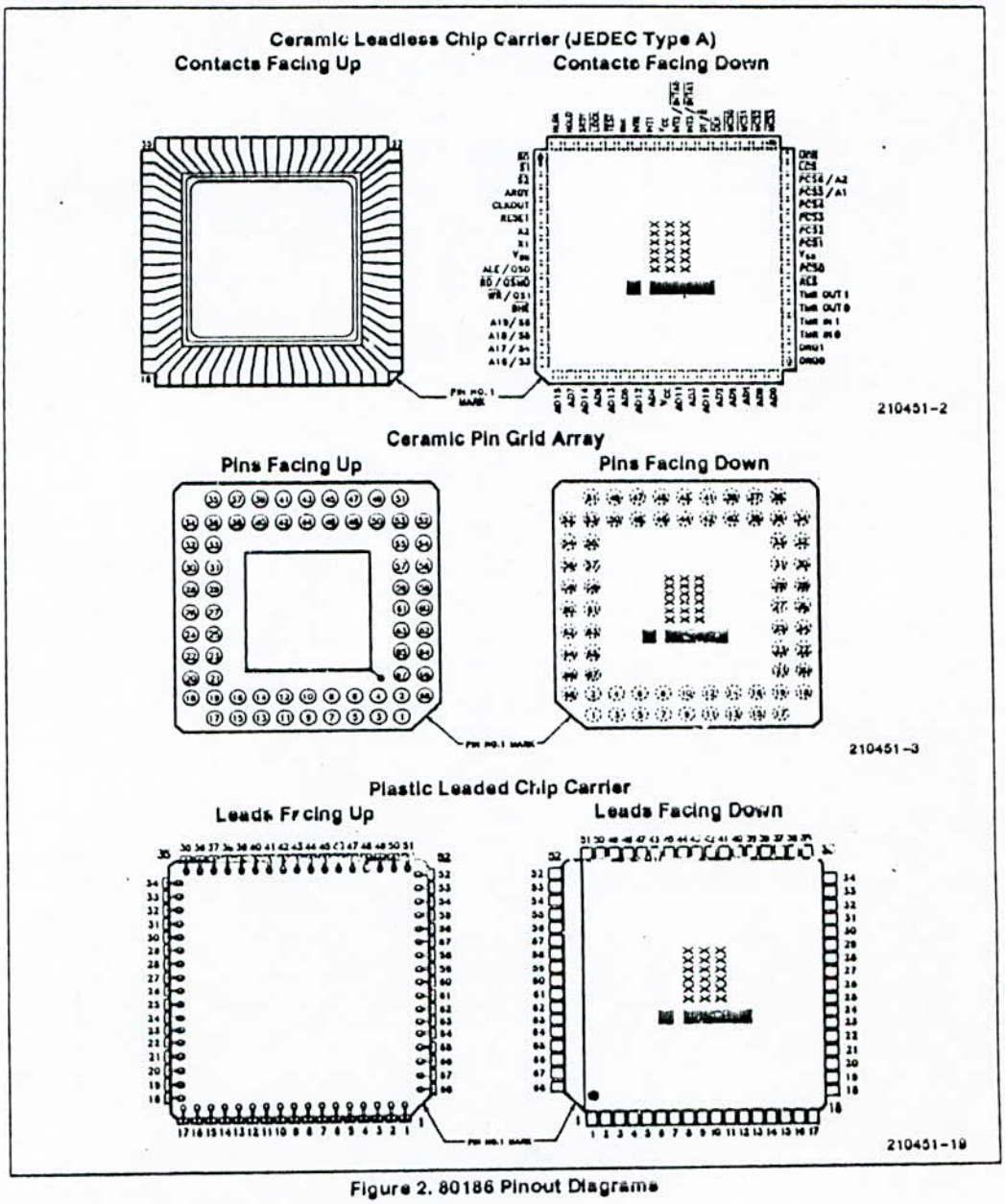


Figure 2. 80186 Pinout Diagrams

80186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	9 43	I	**System Power:** +5 volt power supply.
V_{SS}	26 60	I	System Ground.
RESET	57	O	**Reset** Output indicates that the 80186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the RES signal.
X1 X2	59 58	I O	Crystal Inputs X1 and X2 provide external connections for a fundamental mode parallel resonant crystal for the internal oscillator. Instead of using a crystal, an external clock may be applied to X1 while minimizing stray capacitance on X2. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT.
RES	24	I	An active RES causes the 80186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80186 clock. The 80186 begins fetching instructions approximately 6½ clock cycles after RES is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with RES held LOW. RES is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on RES generation via an RC network.
TEST	47	I/O	TEST is examined by the WAIT instruction. If the TEST input is HIGH when "WAIT" execution begins, instruction execution will suspend. TEST will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80186 is waiting for TEST, interrupts will be serviced. During power-up, active RES is required to configure TEST as an input. This pin is synchronized internally.
TMR IN 0 TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0 TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continuous waveform generation, depending upon the timer mode selected.
DRQ0 DRQ1	18 19	I I	DMA Request is asserted HIGH by an external device when it is ready for DMA Channel 0 or 1 to perform a transfer. These signals are level-triggered and internally synchronized.
NMI	46	I	The Non-Maskable Interrupt input causes a Type 2 interrupt. An NMI transition from LOW to HIGH is latched and synchronized internally, and initiates the interrupt at the next instruction boundary. NMI must be asserted for at least one clock. The Non-Maskable Interrupt cannot be avoided by programming.
INT0 INT1/SELECT INT2/INTA0 INT3/INTA1/IRQ	45 44 42 41	I I I/O I/O	Maskable Interrupt Requests can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When Slave Mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).
RD/QSMD	62	I/O	Read Strobe is an active LOW signal which indicates that the 80186 is performing a memory or I/O read cycle. It is guaranteed not to go LOW before the A/D bus is floated. An internal pull-up ensures that RD is HIGH during RESET. Following RESET the pin is sampled to determine whether the 80186 is to provide ALE, RD, and WR, or queue status information. To enable Queue Status Mode, RD must be connected to GND. RD will float during bus HOLD.
ARDY	55	I	Asynchronous Ready informs the 80186 that the addressed memory space or I/O device will complete a data transfer. The ARDY pin accepts a rising edge that is asynchronous to CLKOUT, and is active HIGH. The falling edge of ARDY must be synchronized to the 80186 clock. Connecting ARDY HIGH will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the SRDY pin.
SRDY	49	I	Synchronous Ready informs the 80186 that the addressed memory space or I/O device will complete a data transfer. The SRDY pin accepts an active-HIGH input synchronized to CLKOUT. The use of SRDY allows a relaxed system timing over ARDY. This is accomplished by elimination of the one-half clock cycle required to internally synchronize the ARDY input signal. Connecting SRDY high will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the ARDY pin.

80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD HLDA	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80186 clock. The 80186 will issue a HLDA (HIGH) in response to a HOLD request at the end of T_4 or T_i. Simultaneous with the issuance of HLDA, the 80186 will float the local bus and control lines. After HOLD is detected as being LOW, the 80186 will lower HLDA. When the 80186 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$ $\overline{MCS1}$ $\overline{MCS2}$ $\overline{MCS3}$	38 37 36 35	O O O O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$ $\overline{PCS1}$ $\overline{PCS2}$ $\overline{PCS3}$ $\overline{PCS4}$	25 27 28 29 30	O O O O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
PCS5/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating PCS5 is software-programmable. PCS5/A1 does not float during bus HOLD. When programmed to provide latched A1, this pin will retain the previously latched value during HOLD.
PCS6/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating PCS6 is software programmable. PCS6/A2 does not float during bus HOLD. When programmed to provide latched A2, this pin will retain the previously latched value during HOLD.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through an external data bus transceiver. When LOW, data is transferred to the 80186. When HIGH the 80186 places write data on the data bus.
$\overline{DEN}$	39	O	Data Enable is provided as a data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state. During RESET, $\overline{DEN}$ is driven HIGH for one clock, then floated. $\overline{DEN}$ also floats during HOLD.
$\overline{LOCK}$	48	O	$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of that instruction. No instruction prefetching will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is driven HIGH for one clock during RESET and then floated.
$\overline{S0}$ $\overline{S1}$ $\overline{S2}$	52 53 54	O O O	Bus cycle status $\overline{S0}$–$\overline{S2}$ are encoded to provide bus-transaction information:

80186 Bus Cycle Status Information

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

The status pins float during HOLD.
$\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{ST}$ as a DT/$\overline{R}$ indicator.

80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
A19/S6 A18/S5 A17/S4 A16/S3	65 66 67 68	O O O O	Address Bus Outputs (16–19) and Bus Cycle Status (3–6) indicate the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, the S6 pin is LOW to indicate a CPU-initiated bus cycle or HIGH to indicate a DMA-initiated bus cycle. During the same T-states, S3, S4, and S5 are always LOW. The status pins float during bus HOLD or RESET.
AD15 AD14 AD13 AD12 AD11 AD10 AD9 AD8 AD7 AD6 AD5 AD4 AD3 AD2 AD1 AD0	1 3 5 7 10 12 14 16 2 4 6 8 11 13 15 17	I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O	Address/Data Bus (0–15) signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7 through D_0. It is LOW during T_1 when a byte is to be transferred onto the lower portion of the bus in memory or I/O operations.
$\overline{BHE}$/S7	64	O	During T_1 the Bus High Enable signal should be used to determine if data is to be enabled onto the most significant half of the data bus; pins D_{15}–D_8. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the higher half of the bus. The S_7 status information is available during T_2, T_3, and T_4. S_7 is logically equivalent to $\overline{BHE}$. $\overline{BHE}$/S7 floats during HOLD.

$\overline{BHE}$ and A0 Encodings

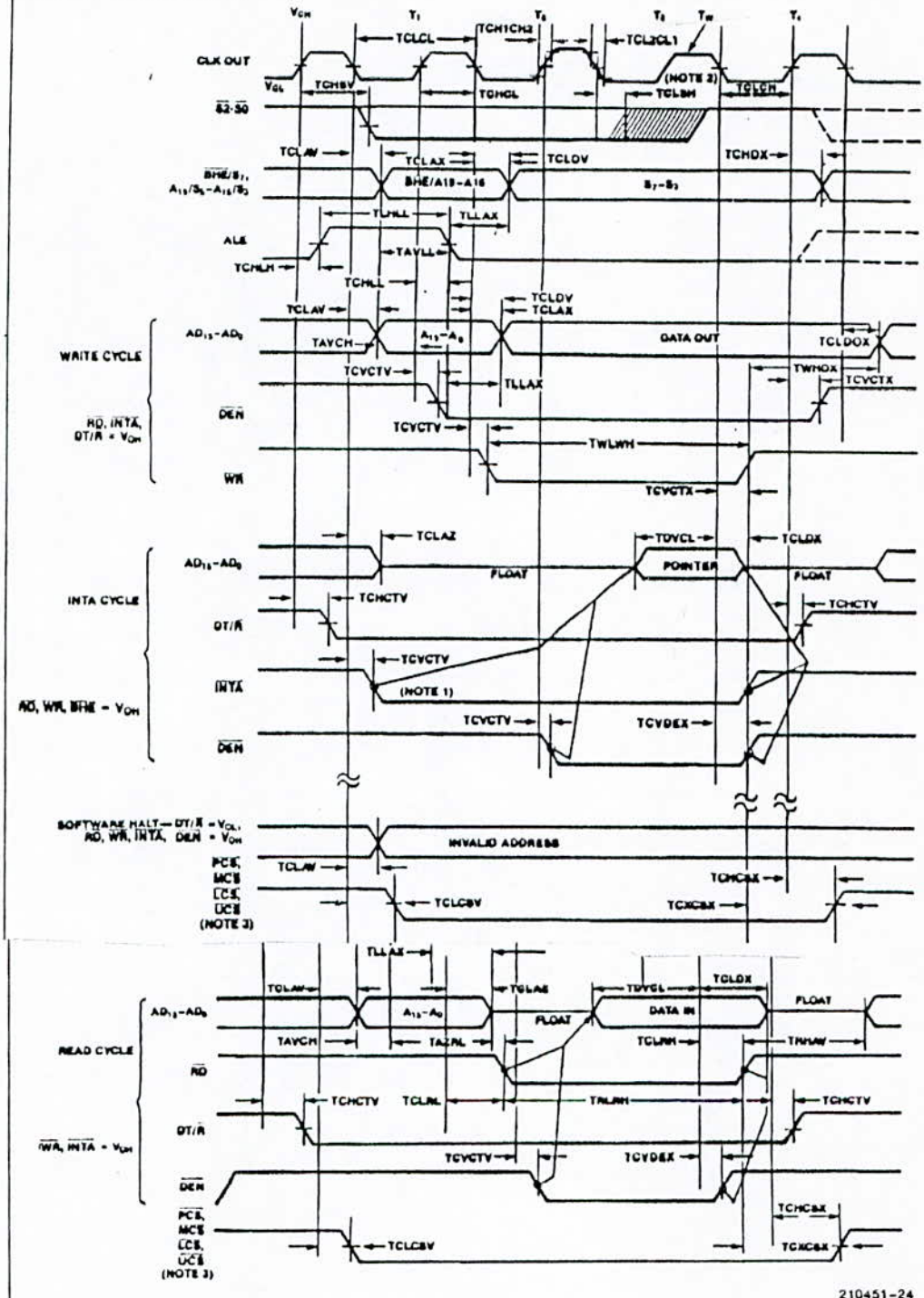
$\overline{BHE}$ Value	A0 Value	Function
0	0	Word Transfer
0	1	Byte Transfer on upper half of data bus (D15–D8)
1	0	Byte Transfer on lower half of data bus (D_7–D_0)
1	1	Reserved

ALE/QS0	61	O	Address Latch Enable/Queue Status 0 is provided by the 80186 to latch the address. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the 8086. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8086. Note that ALE is never floated.
$\overline{WR}$/QS1	63	O	Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during HOLD. When the 80186 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.

QS1	QS0	Queue Operation
0	0	No queue operation
0	1	First opcode byte fetched from the queue
1	1	Subsequent byte fetched from the queue
1	0	Empty the queue

WAVEFORMS

MAJOR CYCLE TIMING

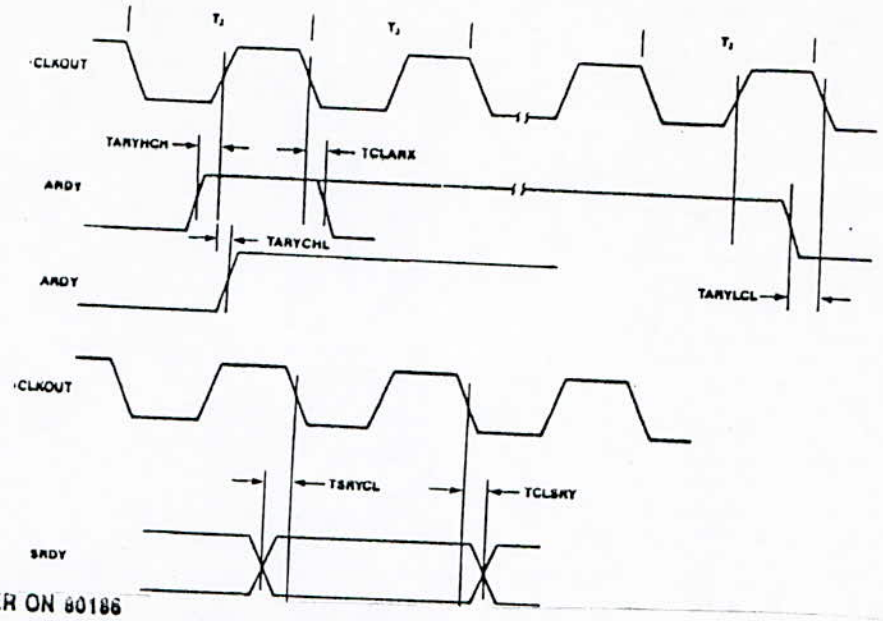


210451-24

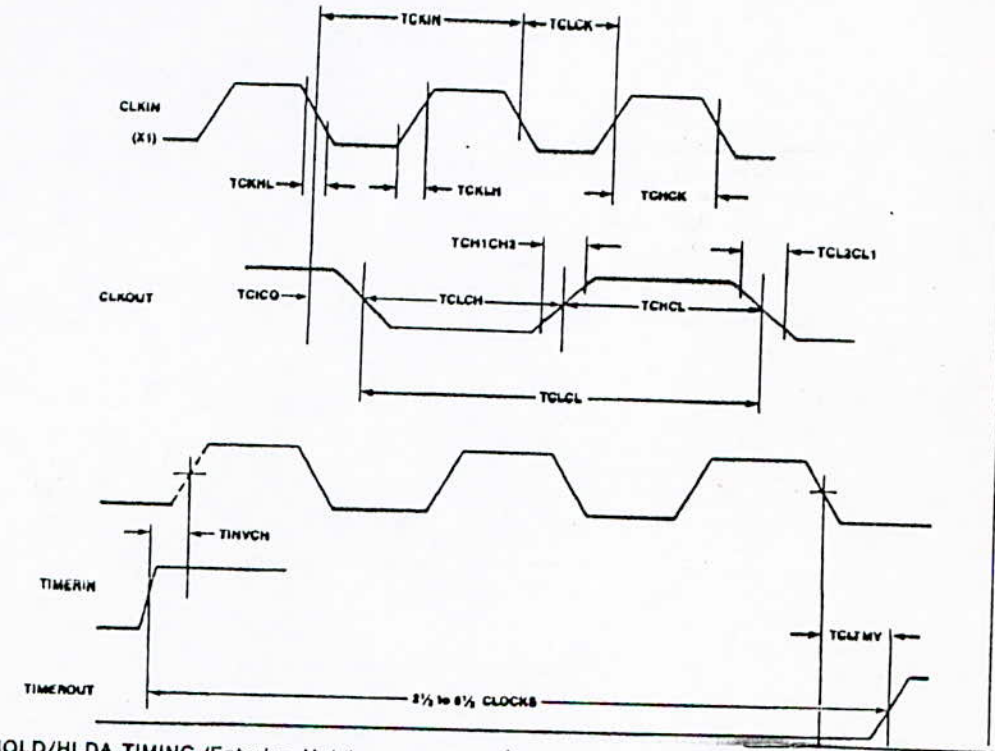
NOTES:
1. INTA occurs one clock later in slave mode.
2. Status inactive just prior to T_4.
3. If latched A1 and A2 are selected instead of PCS5 and PCS6, only T_{CLCSV} is applicable.

WAVEFORMS (Continued)

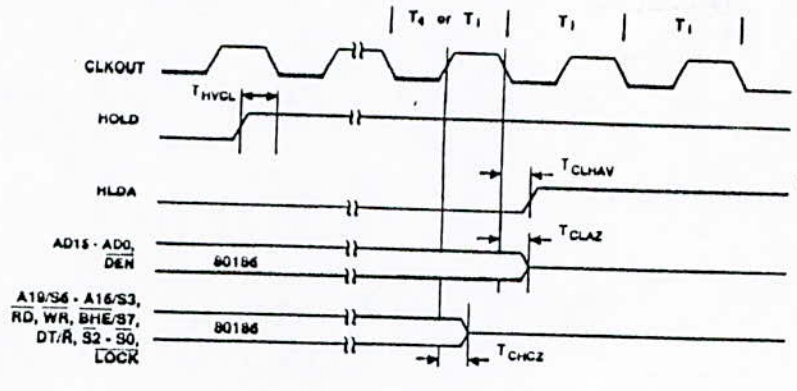
READY TIMING

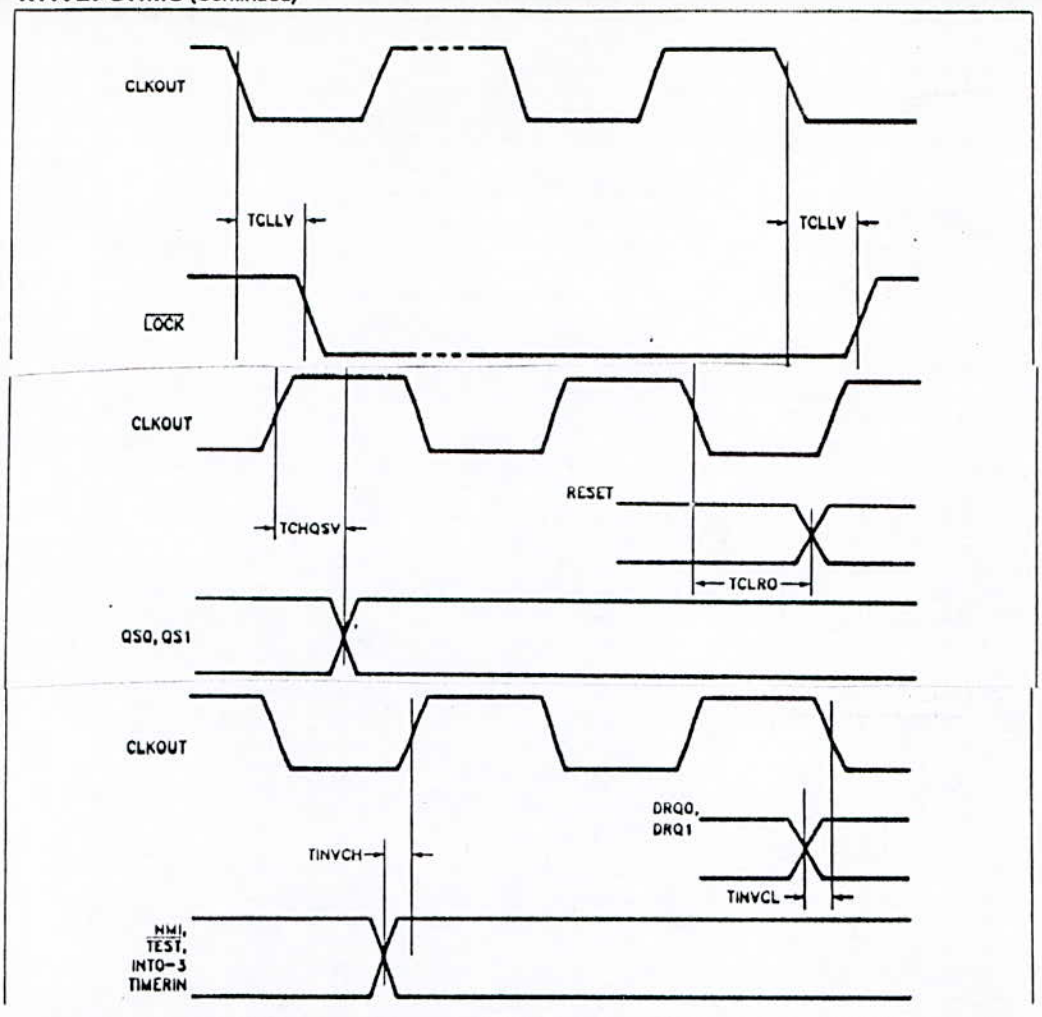


TIMER ON 80186

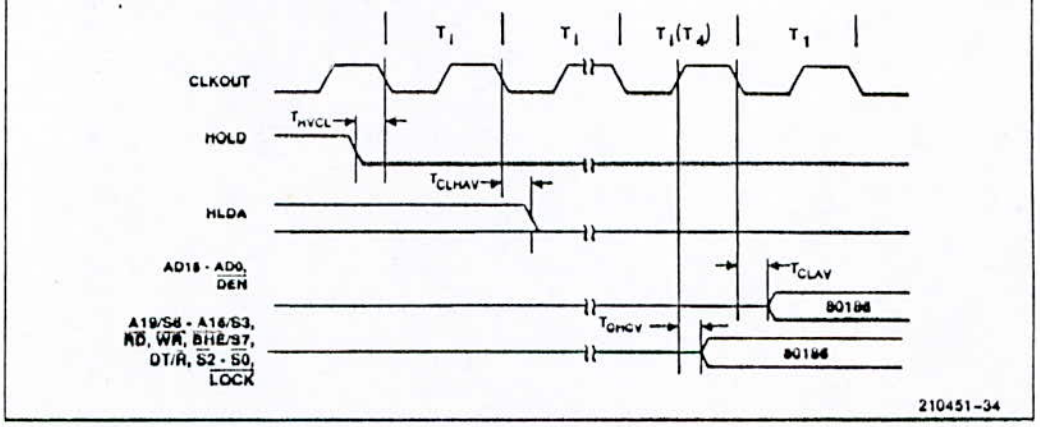


HOLD/HLDA TIMING (Entering Hold)





HOLD/HLDA TIMING (Leaving Hold)



210451-34

→ L'UART AY-3-1015 et LE GBR AY-5-8116.

^ BROCHAGES.

GENERAL INSTRUMENT

AY-3-1015D

UAR/T: Universal Asynchronous Receiver/Transmitter

FEATURES

- DTL and TTL compatible—no interfacing circuits required—drives one TTL load
- Fully Double Buffered—eliminates need for system synchronization, facilitates high-speed operation
- Full Duplex Operation—can handle multiple bauds (receiving-transmitting) simultaneously
- Start Bit Verification—decreases error rate with center sampling
- Receiver center sampling of serial input; 46% distortion immunity
- High Speed Operation
- Three-State Outputs—bus structure capability
- Low Power—minimum power requirements
- Input Protected—eliminates handling problems

AY-3-1015D

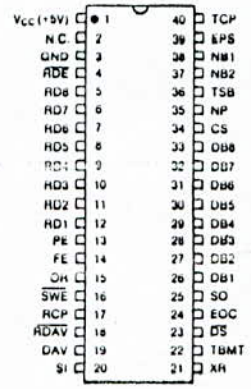
- Single Supply Operation: +4.75V to +5.25V
- 1½ stop bit mode
- External reset of all registers except control bits register
- N-channel Ion Implant Process
- 0 to 25K baud
- Pull-up resistors to V_{CC} on all inputs

DESCRIPTION

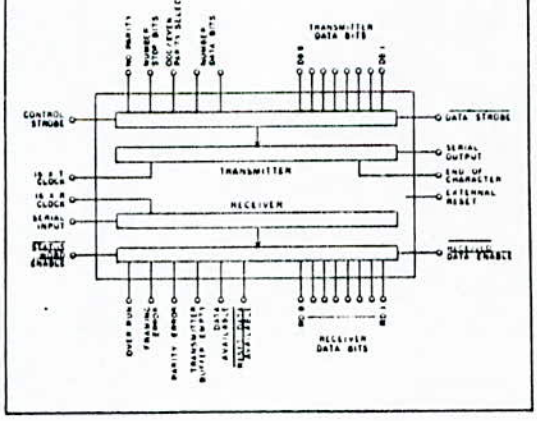
The Universal Asynchronous Receiver/Transmitter (UAR/T) is an LSI subsystem which accepts binary characters from either a terminal device or a computer and receives/transmits this character with appended control and error detecting bits. All characters contain a start bit, 5 to 8 data bits, 1, 1½, or 2 stop bit capability, and either odd/even parity or no parity. In order to make the UAR/T universal, the baud, bits per word, parity mode, and the number of stop bits are externally selectable. The device is constructed on a single monolithic chip. All inputs and outputs are directly compatible with MTOS/MTNS logic, and also with TTL/DTL/CMOS logic without the need for interfacing components. All strobed outputs are three-state logic.

PIN CONFIGURATION
40 LEAD DUAL IN LINE

Top View



BLOCK DIAGRAM



PIN FUNCTIONS

Pin No.	Name (Symbol)	Function
1	Vcc Power Supply (V_{CC})	+5V Supply
2	N.C.	(Not connected)
3	Ground	Ground
4	Received Data Enable ($\overline{RDE}$)	A logic "0" on the receiver enable line places the received data onto the output lines.
5-12	Received Data Bits (RD8-RD1)	These are the 8 data output lines. Received characters are right justified: the LSB always appears on RD1. These lines have tri-state outputs; i.e., they have the normal TTL ouput characteristics when $\overline{RDE}$ is "0" and a high impedance state when $\overline{RDE}$ is "1". Thus, the data output lines can be bus structure oriented.
13	Parity Error (PE)	This line goes to a logic "1" if the received character parity does not agree with the selected parity. Tri-state.
14	Framing Error (FE)	This line goes to a logic "1" if the received character has no valid stop bit. Tri-state.
15	Over-Run (OR)	This line goes to a logic "1" if the previously received character is not read (DAV line not reset) before the present character is transferred to the receiver holding register. Tri-state.
16	Status Word Enable ($\overline{SWE}$)	A logic "0" on this line places the status word bits (PE, FE, OR, DAV, TBMT) onto the output lines. Tri-state.
17	Receiver Clock (RCP)	This line will contain a clock whose frequency is 16 times (16X) the desired receiver baud.
18	Reset Data Available ($\overline{RDAV}$)	A logic "0" will reset the DAV line. The DAV F/F is only thing that is reset.
19	Data Available (DAV)	This line goes to a logic "1" when an entire character has been received and transferred to the receiver holding register. Tri-state. Fig. 8.
20	Serial Input (SI)	This line accepts the serial bit input stream. A Marking (logic "1") to spacing (logic "0") transition is required for initiation of data reception. Fig. 7, 8.
21	External Reset (XR)	Resets all registers. Sets SO, EOC, and TBMT to a logic "1". Resets DAV, and error flags to "0". Clears input data buffer. Must be tied to logic "0" when not in use.
22	Transmitter Buffer Empty (TBMT)	The transmitter buffer empty flag goes to a logic "1" when the data bits holding register may be loaded with another character. Tri-state. See Fig. 14, 16.
23	Data Strobe ($\overline{DS}$)	A strobe on this line will enter the data bits into the data bits holding register. Initial data transmission is initiated by the rising edge of $\overline{DS}$. Data must be stable during entire strobe.
24	End of Character (EOC)	This line goes to a logic "1" each time a full character is transmitted. It remains at this level until the start of transmission of the next character. See Fig. 13, 15.
25	Serial Output (SO)	This line will serially, by bit, provide the entire transmitted character. It will remain at a logic "1" when no data is being transmitted.
26-33	Data Bit Inputs (DB1-DB8)	There are up to 8 data bit input lines available.
34	Control Strobe (CS)	A logic "1" on this lead will enter the control bits (EPS, NB1, NB2, TSB, NP) into the control bits holding register. This line can be strobed or hard wired to a logic "1" level.
35	No Parity (NP)	A logic "1" on this lead will eliminate the parity bit from the transmitted and received character (no PE indication). The stop bit(s) will immediately follow the last data bit. If not used, this lead must be tied to a logic "0".
36	Number of Stop Bits (TSB)	This lead will select the number of stop bits, 1 or 2, to be appended immediately after the parity bit. A logic "0" will insert 1 stop bit and a logic "1" will insert 2 stop bits. The combined selection of 2 stop bits and 5 bits/character will produce 1½ stop bits.
37-38	Number of Bits/Character (NB2, NB1)	These two leads will be internally decoded to select either 5, 6, 7 or 8 data bits/character. NB2 NB1 Bits/Character 0 0 5 0 1 6 1 0 7 1 1 8
39	Odd/Even Parity Select (EPS)	The logic level on this pin selects the type of parity which will be appended immediately after the data bits. It also determines the parity that will be checked by the receiver. A logic "0" will insert odd parity and a logic "1" will insert even parity.
40	Transmitter Clock (TCP)	This line will contain a clock whose frequency is 16 times (16X) the desired transmitter baud

GENERAL INSTRUMENT

AY-5-8116/8116T
AY-5-8136/8136T

Dual Baud Rate Generator

FEATURES

- Single +5V power supply
- On-chip crystal oscillator 8116/8136 or external freq. input 8116/8116T/8136/8136T
- Direct compatibility with UART/USRT
- Dual selectable 16x clock outputs
- High freq. reference output (Available only on 8136/8136T)
- Reprogrammable ROM allowing generation of non-standard frequencies
- TTL, MOS compatibility
- Pin for pin and functionally compatible with SMC's COM8116/8116T/8136/8136T
- General Instrument Advanced N-Channel Silicon Gate Process

DESCRIPTION

The General Instrument AY-5-8116/8136 Series is a very versatile family of Dual Baud Rate Generators. The AY-5-8116/8116T and AY-5-8136/8136T are pin for pin and functionally equivalent to SMC's COM8116/8116T/8136/8136T, respectively.

The AY-5-8116/8136 is designed to generate the full spectrum of 16 asynchronous/synchronous data communication frequencies for use with 16X UART/USRT devices.

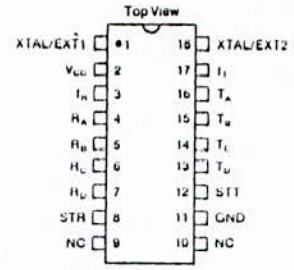
An on-chip crystal oscillator available on the 8116 and 8136 is capable of providing a master reference frequency. Alternatively, complimentary TTL level clock signals can be input to pins 1 and 18. The 8116T and 8136T are only suitable for this external TTL reference. When using TTL outputs to drive the XTAL/EXT inputs, they should not be used to drive other TTL inputs due to excessive loading which may result in a reduction of noise immunity.

Dividers are used on the output of the oscillator/buffer which generate the output frequencies f_T and f_H. These dividers can divide any integer from 6 to $2^{19}+1$, inclusive. When using an even divisor, the output will be square; an odd divisor will cause the output to be high longer than it is low by one clock period (f_X). The clock frequency (f_X) is used by the 8136/8136T to provide a high frequency output ($f_X/4$).

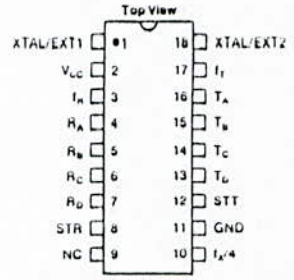
The 8116/8136 family allows generation of other frequencies with the use of its two divisor ROMs which contain 16 divisors, each 19 bits wide, allowing for up to 32 different divisors on custom parts.

PIN CONFIGURATIONS

AY-5-8116/8116T



AY-5-8136/8136T

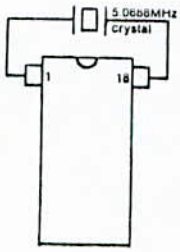


Externally strobed data latches are used to hold the divisor select bits, R_A-R_D and T_A-T_D. The strobe inputs, STR or STT, allow data to pass directly through the data latch when in the high state. A new frequency is initiated within 3.5 usec of a change in any of the four divisor select bits read by the device. Pull-up resistors are provided on the divisor select inputs while are not present on the strobe inputs.

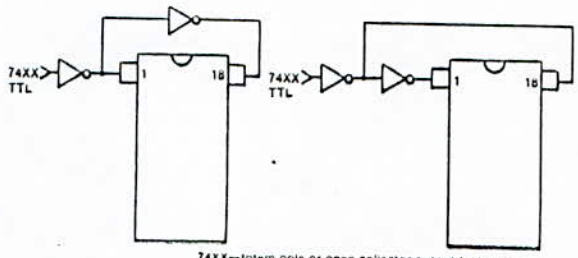
PIN FUNCTIONS

Pin No.	Signal	Function
1	XTAL/EXT1	Input is either one pin of the crystal package or one polarity of the external input.
2	V_{cc}	Positive power supply — normally +5V.
3	f_H	This output runs at a frequency selected by the Receiver divisor select data bits.
4-7	R_A, R_B, R_C, R_D	These inputs, as shown in Table 1, select the receiver output frequency, f_H.
8	STR	A high level input strobe loads the receiver data (R_A, R_B, R_C, R_D) into the receiver divisor select register. This input may be strobed or hard-wired to a high level.
9	NC	
10	NC or $f_X/4$	NC (8116/8116T), $f_X/4$ (8136/8136T)
11	GND	Ground
12	STT	A high level input strobe loads the transmitter data (T_A, T_B, T_C, T_D) into the transmitter divisor select register. This input may be strobed or hard-wired to a high level.
13-16	T_D, T_C, T_B, T_A	These inputs, as shown in Table 1, select the transmitter output frequency, f_T.
17	f_T	This output runs at a frequency selected by the Transmitter divisor select data bits.
18	XTAL/EXT2	This input is either the other pin of the crystal package or the other polarity of the external input.

CRYSTAL OPERATION AY-5-8116/8136



EXTERNAL INPUT OPERATION AY-5-8116/8116T/8136/8136T



74XX—totem pole or open collector output (external pull-up resistor required)

Output Freq. AY-5-8116/8116T/8136/8136T

REFERENCE FREQUENCY = 5.068800MHz

Divisor Select DCBA	Desired Baud Rate	Clock Factor	Desired Frequency (KHz)	Divisor	Actual Baud Rate	Actual Frequency (KHz)	Deviation
0000	50.00	16X	0.80000	6336	50.00	0.800000	0.0000%
0001	75.00	16X	1.20000	4224	75.00	1.200000	0.0000%
0010	110.00	16X	1.76000	2880	110.00	1.760000	0.0000%
0011	134.50	16X	2.15200	2355	134.52	2.152357	0.0166%
0100	150.00	16X	2.40000	2112	150.00	2.400000	0.0000%
0101	300.00	16X	4.80000	1056	300.00	4.800000	0.0000%
0110	600.00	16X	9.60000	528	600.00	9.600000	0.0000%
0111	1200.00	16X	19.20000	264	1200.00	19.200000	0.0000%
1000	1800.00	16X	28.80000	176	1800.00	28.800000	0.0000%
1001	2000.00	16X	32.00000	158	2005.06	32.081013	0.2532%
1010	2400.00	16X	38.40000	132	2400.00	38.400000	0.0000%
1011	3600.00	16X	57.60000	88	3600.00	57.600000	0.0000%
1100	4800.00	16X	76.80000	66	4800.00	76.800000	0.0000%
1101	7200.00	16X	115.20000	44	7200.00	115.200000	0.0000%
1110	9600.00	16X	153.60000	33	9600.00	153.600000	0.0000%
1111	19200.00	16X	307.20000	16	19800.00	316.800000	3.1250%

Output Freq. AY-5-8116/8116T/8136/8136T-005

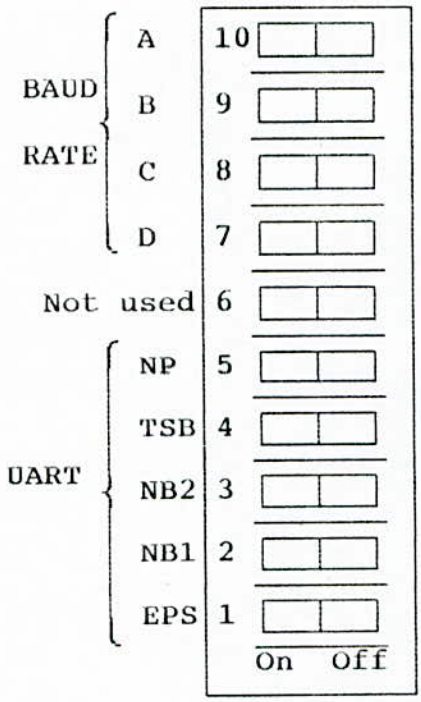
REFERENCE FREQUENCY = 4.915200MHz

Divisor Select DCBA	Desired Baud Rate	Clock Factor	Desired Frequency (KHz)	Divisor	Actual Baud Rate	Actual Frequency (KHz)	Deviation
0000	50.00	16X	0.80000	6144	50.00	0.800000	0.0000%
0001	75.00	16X	1.20000	4096	75.00	1.200000	0.0000%
0010	110.00	16X	1.76000	2793	109.93	1.758983	0.0100%
0011	134.50	16X	2.15200	2284	134.50	2.152000	0.0000%
0100	150.00	16X	2.40000	2048	150.00	2.400000	0.0000%
0101	300.00	16X	4.80000	1024	300.00	4.800000	0.0000%
0110	600.00	16X	9.60000	512	600.00	9.600000	0.0000%
0111	1200.00	16X	19.20000	256	1200.00	19.200000	0.0000%
1000	1800.00	16X	28.80000	171	1796.49	28.743859	0.1949%
1001	2000.00	16X	32.00000	154	1994.81	31.916883	0.2597%
1010	2400.00	16X	38.40000	128	2400.00	32.000000	0.0000%
1011	3600.00	16X	57.60000	85	3614.11	57.825882	0.3921%
1100	4800.00	16X	76.80000	64	4800.00	76.800000	0.0000%
1101	7200.00	16X	115.20000	43	7144.19	114.306976	0.7751%
1110	9600.00	16X	153.60000	32	9600.00	153.600000	0.0000%
1111	19200.00	16X	307.20000	16	19200.00	307.200000	0.0000%

CONFIGURATION DU BAUD RATE ET DE L'UART.

Pour la configuration du Baud Rate et de l'Uart 9 (neuf) microswitchs ont été prévûs. Quatres de ces microswitchs (S10,S9,S8 et S7) servent à selectionner les fréquences d'émission et de reception pour l'UART. Les cinq restants servent à choisir le format de la donnée serielle (nombre de bits stop, parité, nombre de bits/caractère).

Choix de la vitesse



DCBA	Baud Rate
0010	110
0100	150
0101	300
0110	600
0111	1200
1010	2400
1100	4800
1110	9600
1111	19200

Off ⟶ "1"
On ⟶ "0"

Choix du format de la donnée

NP:No Parity. Off ⟶ No parity

TSB: number of Stop Bits.
On ⟶ 1 stop bit;
Off ⟶ 2 stop bits.

NB1,NB2: Number on Bits/caracter.

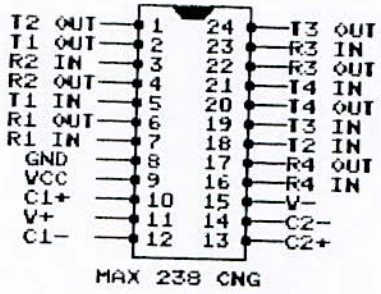
NB2	NB1	Bits/caracter
On	On	5
Off	Off	8
On	Off	6
Off	On	7

EPS: odd/Even Parity Select.
On ⟶ Odd parity;
Off ⟶ Even parity.

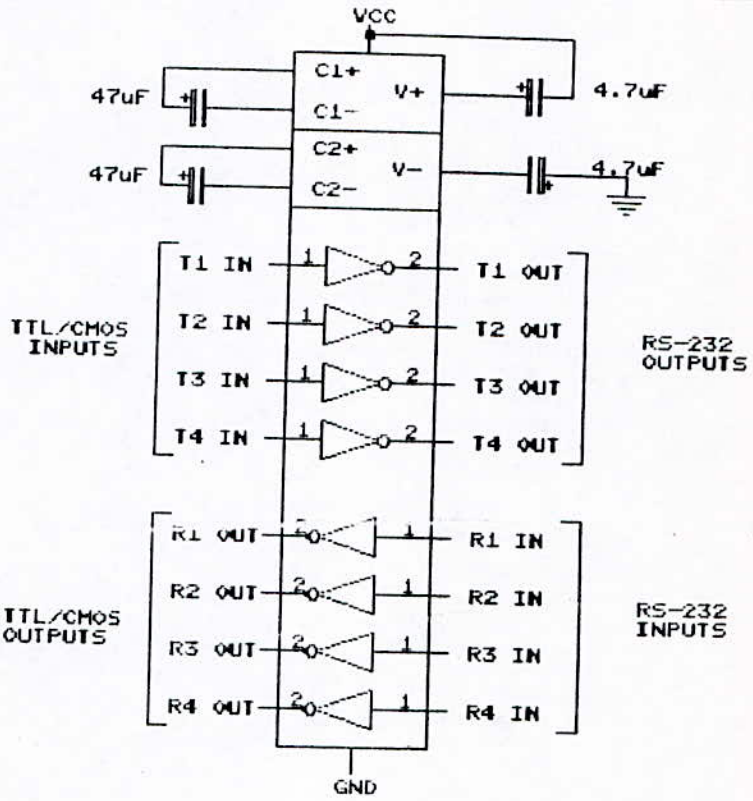
Exemple de configuration: 9600,n,1,8

S10	S9	S8	S7	S6	S5	S4	S3	S2	S1
On	off	off	off	//	off	off	off	On	off

→ Circuit MAX238 CNG.



MAX 238 CNG



GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for TEST pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

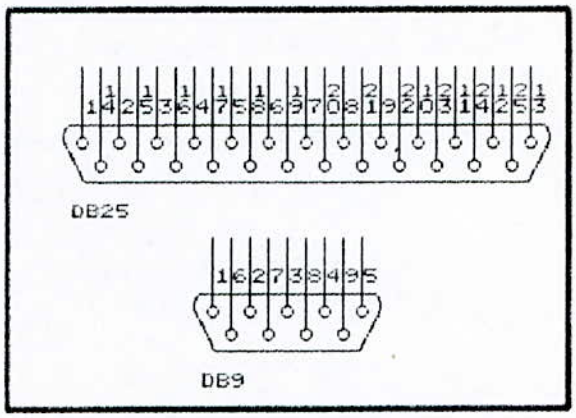
CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

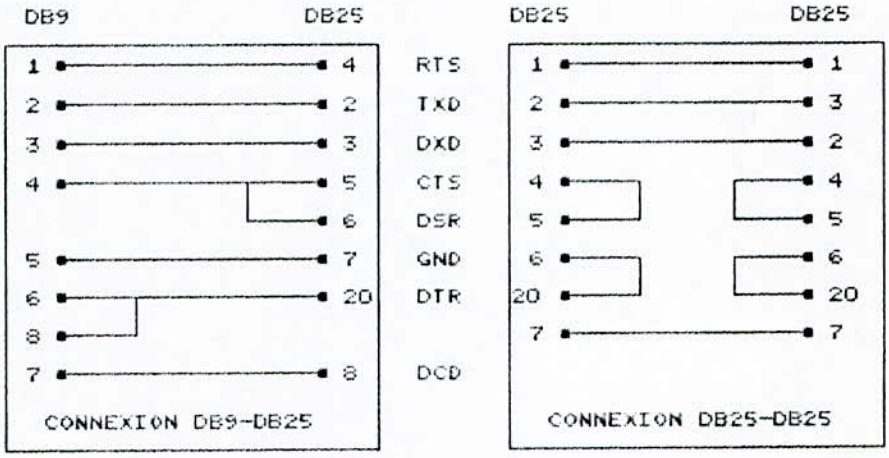
ANNEXE C : LA NORME RS232C :

PINE	NAME	FUNCTION
1	FG	Frame Ground (masse châssis)
2	TD	Transmit Data
3	RD	Receive Data
4	RTS	Request To Send (demande d'émission)
5	CTS	Clear To Send
6	DSR	Data Set Ready (émetteur prêt)
7	SG	Signal Ground (masse des signaux)
8	DCD	Data Carrier Detect (detection porteuse)
9	-	
10	-	
11	-	
12	(S)DCD	Secondary DCD
13	(S)CTS	Secondary CTS
14	(S)TD	SEcondary TD
15	TC	Transmit Clock
16	(S)RD	Secondary RD
17	RC	Receive Clock
18	-	
19	(S)RTS	Secondary RTS
20	DTR	Data Terminal Ready
21	SQ	Signal Quality
22	RI	Ring Indicator
23	-	
24	ETC	External Transmit Clock
25	-	

CONNECTEURS DB25 ET DB9.



CONNECTEURS DB25 ET DB9.



LIAISONS SERIELLES ENTRE DEUX CARTES.

ANNEXE D : DESCRIPTION DU FORMAT INTEL-HEXA.

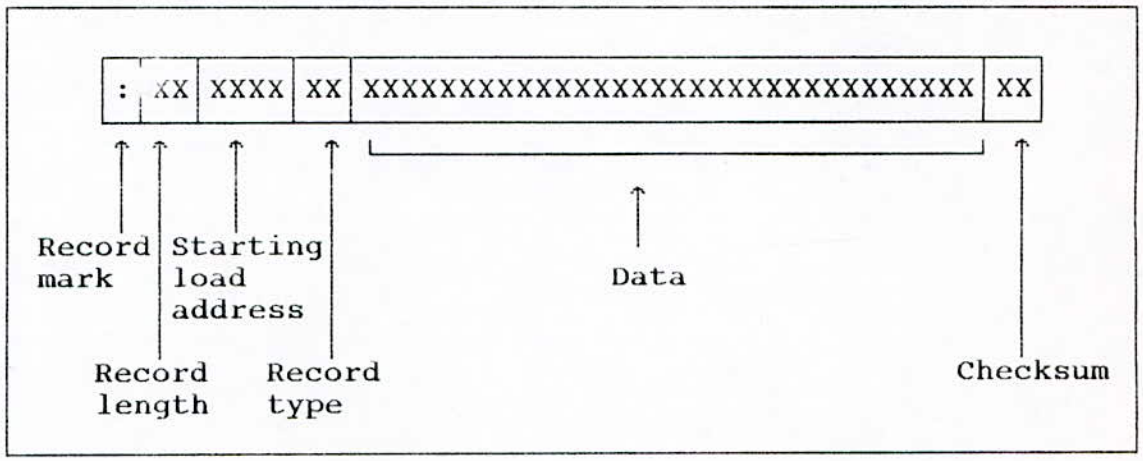
Le format INTEL-HEXA est une methode d'encodage de données ASCII.Le format INTEL-HEXA est formé de chaine de caractères ASCII, appelées enregistrements.

Ce format permet trois types d'enregistrements :

- enregistrements de données;
- enregistrements d'adresse étendue;
- enregistrement de fin de fichier.

Chaque enregistrement débute par le caractère ":"(hexa 3AH) et se termine par CR/LF (Carriage Return/Line Feed). Chaque octet est codé en hexadécimal, et représenté par deux caractères ASCII.

FORMAT D'ENREGISTREMENT DE DONNEES :



Record mark : ":" ASCII;

Record length : nombre d'octets de données contenus dans l'enregistrement (de 00 à FF);

Starting load address : quatre caractères ASCII (pour deux octets) spécifiant l'adresse à laquelle doivent rangées les données;

Record type : deux caractères spécifient le type d'enregistrement 00, 01, 02;

Data : données en héxadécimal sur deux caractères ASCII;

Checksum : un octet héxa, codé sur deux caractères ASCII, complément à deux de la somme sur huit bits des octets de l'enregistrement jusqu'au champ de données inclus.

Les types d'enregistrement sont les suivants :

Type 00 : Enregistrement de données;

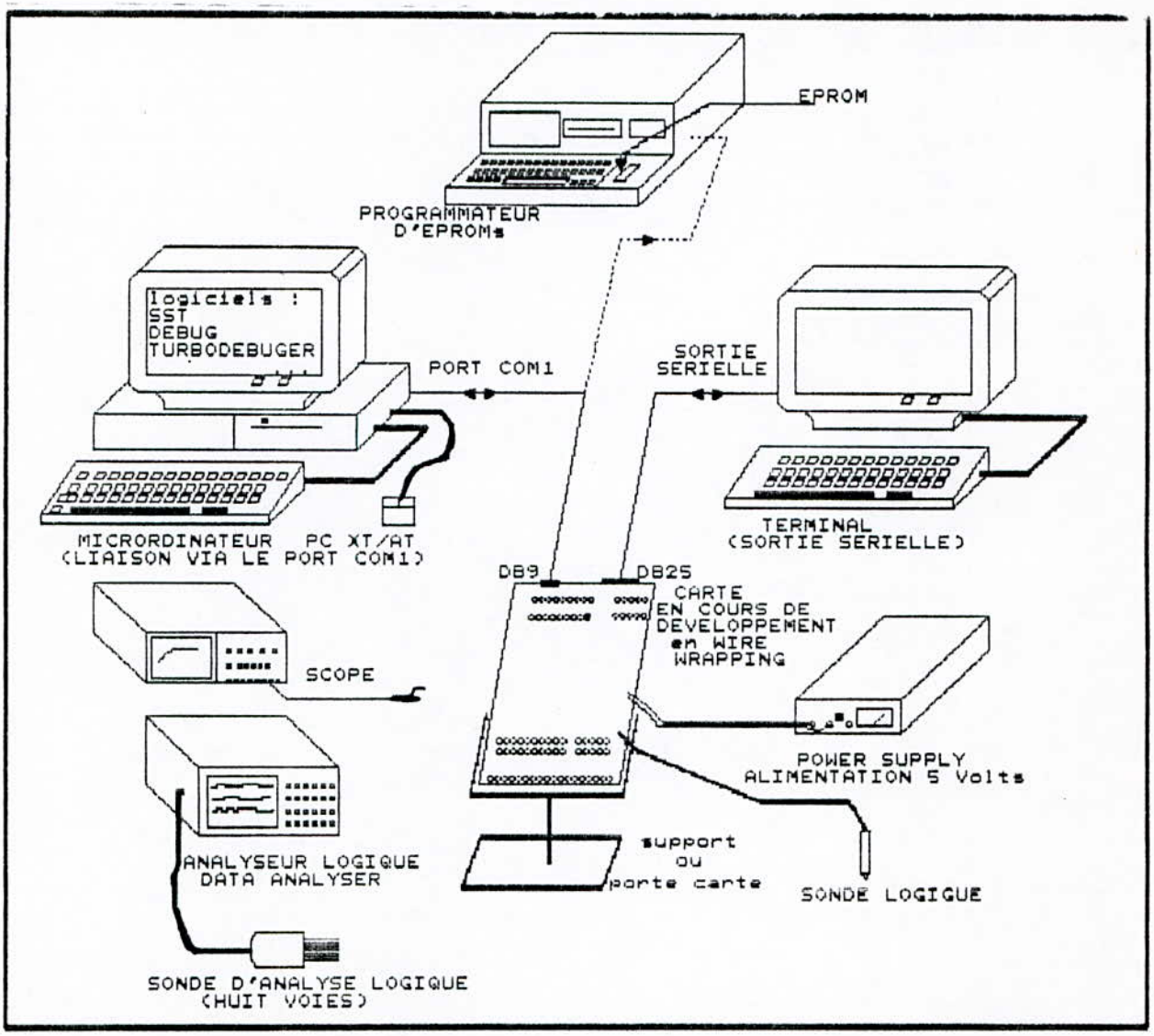
Type 01 : Enregistrement de fin de fichier. Dans ce type d'enregistrement, il n'y a pas de champ de données.

Type 02 : Enregistrement d'adresse étendue. Il n' y a pas de champ de donnée. Le champ adresse contient les bits 4 à 20 de l'adresse à laquelle les octets suivants devront etre chargés. L'adresse contenue dans les enregistrements de type 00 suivants sera ajoutée à l'adresse étendue reçue dans l'enregistrement de type 02.

ex:

-Enregistremnt de données:
:10000000064010FE21632F360021002F22032F31DE

- Enregistremenr de fin de fichier:
:00000001FF


EPROM
PROGRAMMATEUR
D'EPROMs
logiciels :
SST
DEBUG
TURBODEBUGER
PORT COM1
SORTIE
SERIELLE
MICRORDINATEUR PC XT/AT
(LIAISON VIA LE PORT COM1)
TERMINAL
(SORTIE SERIELLE)
DB9
DB25
CARTE
EN COURS DE
DEVELOPPEMENT
en WIRE
WRAPPING
SCOPE
POWER SUPPLY
ALIMENTATION 5 Volts
support
ou
porte carte
ANALYSEUR LOGIQUE
DATA ANALYSER
SONDE LOGIQUE
SONDE D'ANALYSE LOGIQUE
(HUIT VOIES)

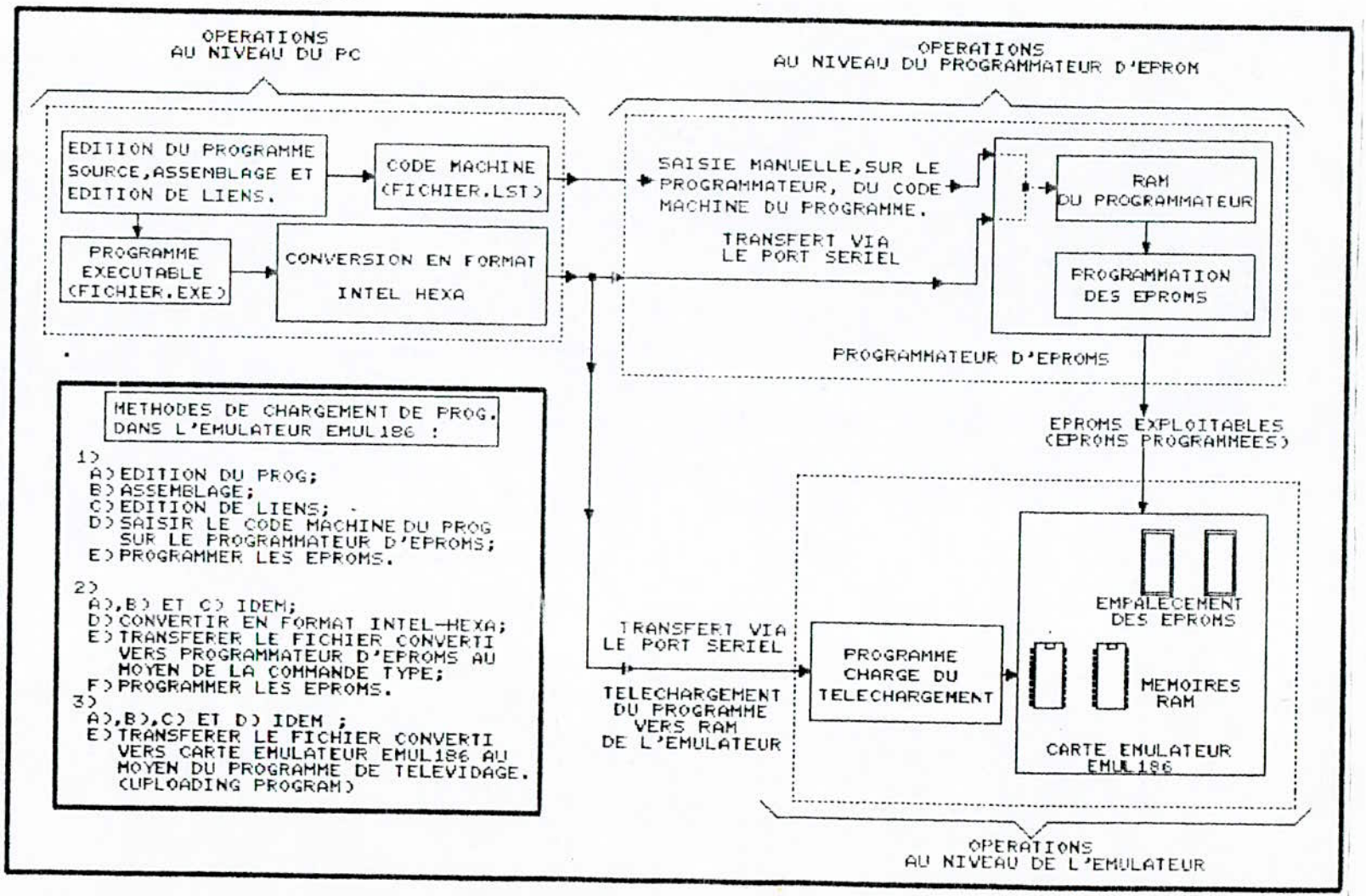
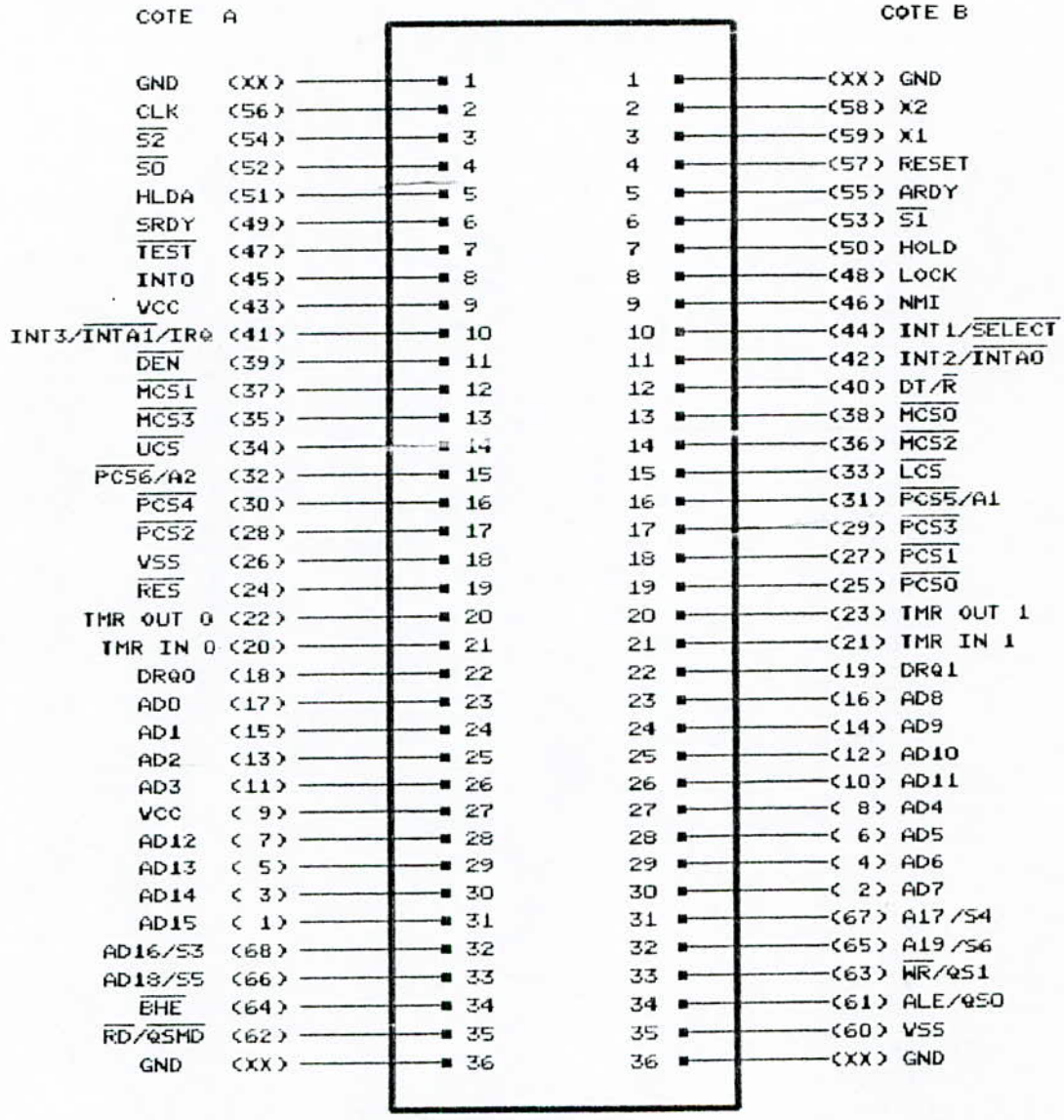

OPERATIONS
AU NIVEAU DU PC
EDITION DU PROGRAMME SOURCE,ASSEMBLAGE ET EDITION DE LIENS.
CODE MACHINE (FICHIER.LST)
PROGRAMME EXECUTABLE (FICHIER.EXE)
CONVERSION EN FORMAT INTEL HEXA
OPERATIONS
AU NIVEAU DU PROGRAMMATEUR D'EPROM
SAISIE MANUELLE,SUR LE PROGRAMMATEUR, DU CODE MACHINE DU PROGRAMME.
TRANSFERT VIA LE PORT SERIEL
RAM DU PROGRAMMATEUR
PROGRAMMATION DES EPROMS
PROGRAMMATEUR D'EPROMS
METHODES DE CHARGEMENT DE PROG. DANS L'EMULATEUR EMUL186 :
1)
A) EDITION DU PROG;
B) ASSEMBLAGE;
C) EDITION DE LIENS;
D) SAISIR LE CODE MACHINE DU PROG SUR LE PROGRAMMATEUR D'EPROMS;
E) PROGRAMMER LES EPROMS.
2)
A),B) ET C) IDEM;
D) CONVERTIR EN FORMAT INTEL-HEXA;
E) TRANSFERER LE FICHIER CONVERTI VERS PROGRAMMATEUR D'EPROMS AU MOYEN DE LA COMMANDE TYPE;
F) PROGRAMMER LES EPROMS.
3)
A),B),C) ET D) IDEM ;
E) TRANSFERER LE FICHIER CONVERTI VERS CARTE EMULATEUR EMUL186 AU MOYEN DU PROGRAMME DE TELEVIDAGE. (UPLOADING PROGRAM)
EPROMS EXPLOITABLES (EPROMS PROGRAMMEES)
EMPALECEMENT DES EPROMS
TRANSFERT VIA LE PORT SERIEL
PROGRAMME CHARGE DU TELECHARGEMENT
MEMOIRES RAM
TELECHARGEMENT DU PROGRAMME VERS RAM DE L'EMULATEUR
CARTE EMULATEUR EMUL186
OPERATIONS
AU NIVEAU DE L'EMULATEUR

FIGURE: CORRESPONDANCE DES PINES DU CONNECTEUR DE LA CARTE FILLE AUX PINES DU MICROCONTROLEUR i80186.



NOTE: Les pines referencees par (XX) n'appartiennent pas au processeur. (les pines 1 et 36 de chaque cote du connecteur).

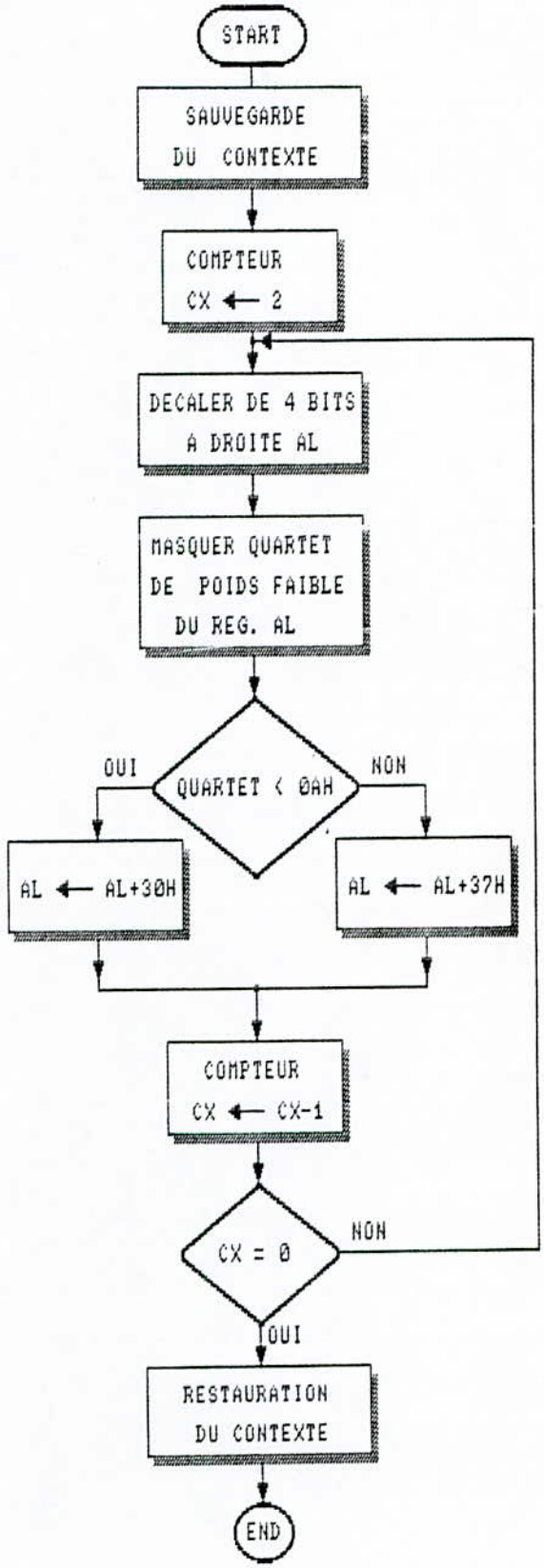
Annexe H :

CODE ASCII — IEEE

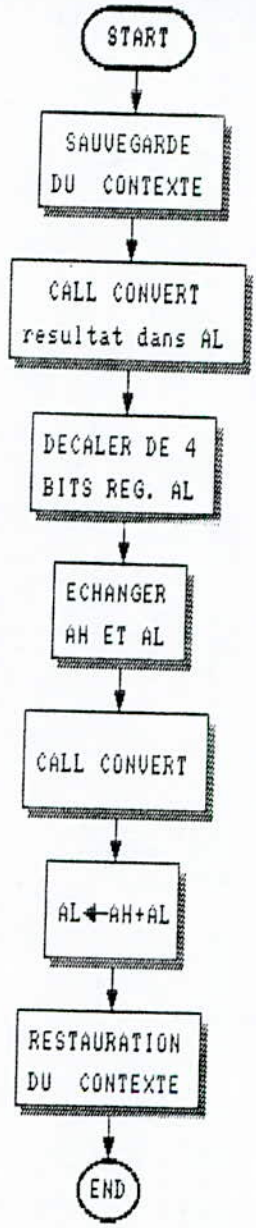
B7 B6 B5 / B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTRÔLE		CHIFFRE MAJUSCULE		LETTRE MAJUSCULE		LETTRE MINUSCULE	
0 0 0 0	0 NUL 0	10 DLE 16	20 SP 32	30 0 48	40 @ 64	50 P 80	60 ` 96	70 p 112
0 0 0 1	1 GTL SOH 1	11 LLO DC1 17	21 ! 33	31 1 49	41 A 65	51 Q 81	61 a 97	71 q 113
0 0 1 0	2 STX 2	12 DC2 18	22 " 34	32 2 50	42 B 66	52 R 82	62 b 98	72 r 114
0 0 1 1	3 ETX 3	13 DC3 19	23 # 35	33 3 51	43 C 67	53 S 83	63 c 99	73 s 115
0 1 0 0	4 SDC EOT 4	14 DCL DC4 20	24 $ 36	34 4 52	44 D 68	54 T 84	64 d 100	74 t 116
0 1 0 1	5 PPC ENQ 5	15 PPU NAK 21	25 % 37	35 5 53	45 E 69	55 U 85	65 e 101	75 u 117
0 1 1 0	6 ACK 6	16 SYN 22	26 & 38	36 6 54	46 F 70	56 V 86	66 f 102	76 v 118
0 1 1 1	7 BEL 7	17 ETB 23	27 ' 39	37 7 55	47 G 71	57 W 87	67 g 103	77 w 119
1 0 0 0	8 GET BS 8	18 SPE CAN 24	28 (40	38 8 56	48 H 72	58 X 88	68 h 104	78 x 120
1 0 0 1	9 TCT HT 9	19 SPD EM 25	29) 41	39 9 57	49 I 73	59 Y 89	69 i 105	79 y 121
1 0 1 0	A LF 10	1A SUB 26	2A * 42	3A : 58	4A J 74	5A Z 90	6A j 106	7A z 122
1 0 1 1	B VT 11	1B ESC 27	2B + 43	3B ; 59	4B K 75	5B [91	6B k 107	7B { 123
1 1 0 0	C FF 12	1C FS 28	2C , 44	3C < 60	4C L 76	5C \ 92	6C l 108	7C | 124
1 1 0 1	D CR 13	1D GS 29	2D - 45	3D = 61	4D M 77	5D] 93	6D m 109	7D } 125
1 1 1 0	E SO 14	1E RS 30	2E . 46	3E > 62	4E N 78	5E ^ 94	6E n 110	7E ~ 126
1 1 1 1	F SI 15	1F US 31	2F / 47	3F UNL ? 63	4F O 79	5F UNT _ 95	6F o 111	7F DEL 127
	GROUPE DE COMMANDE ADRES. (ACG)	UNIV. (UCG)	GROUPE D'ADRESSE ECOUTEUR (LAG)		GROUPE D'ADRESSE PARLEUR (TAG)		GROUPE DE COMMANDE SECONDAIRE (SCG)	
	GROUPE DE COMMANDE PRIMAIRE (PCG)							

Hexa / Deci — 1 ABC XY 1 — IEEE

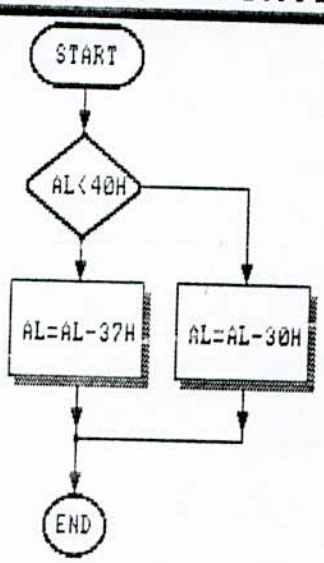
Procedure Hexa_to_Ascii



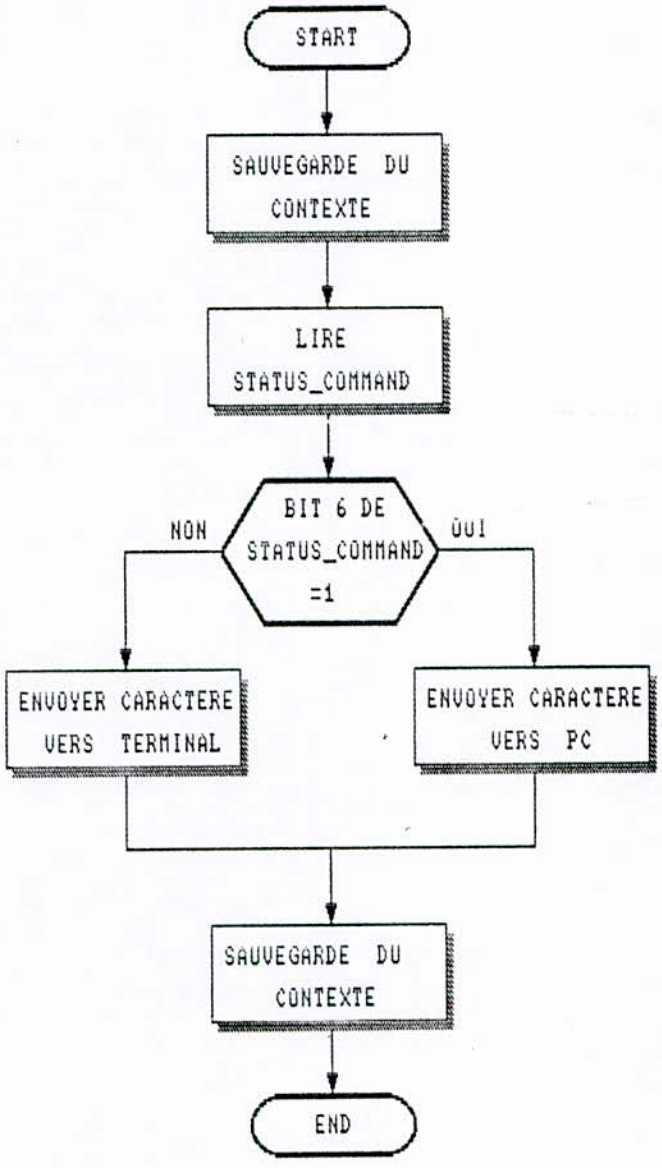
Procedure Ascii_to_Hexa.



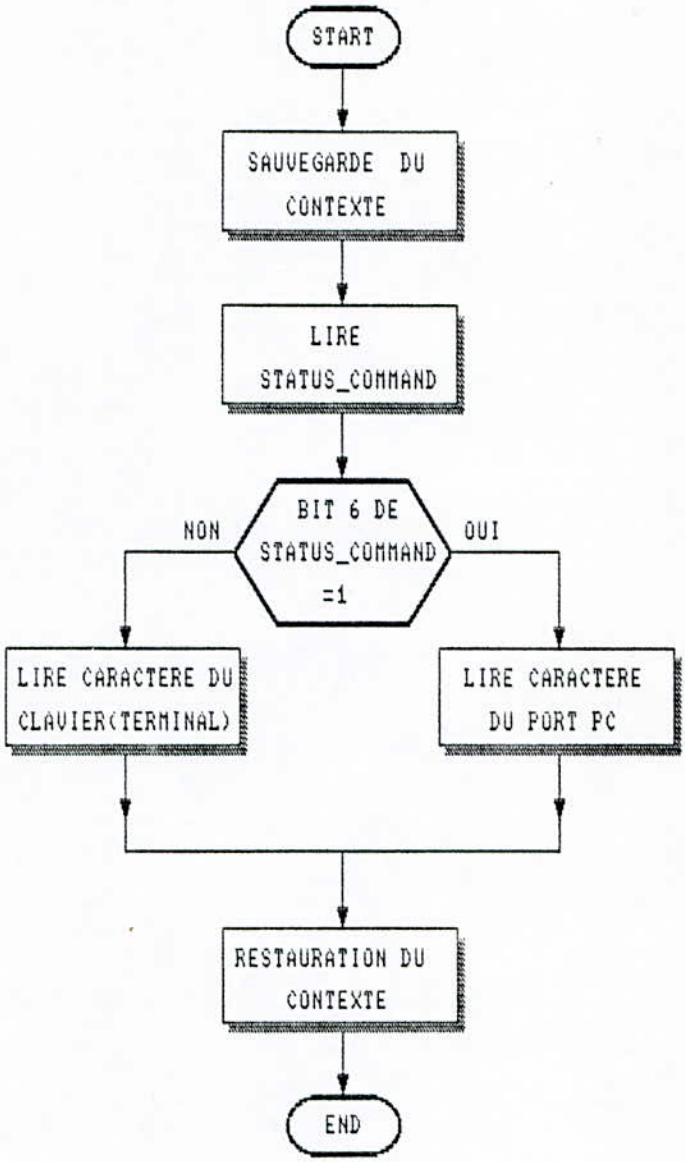
Procedure CONVERT



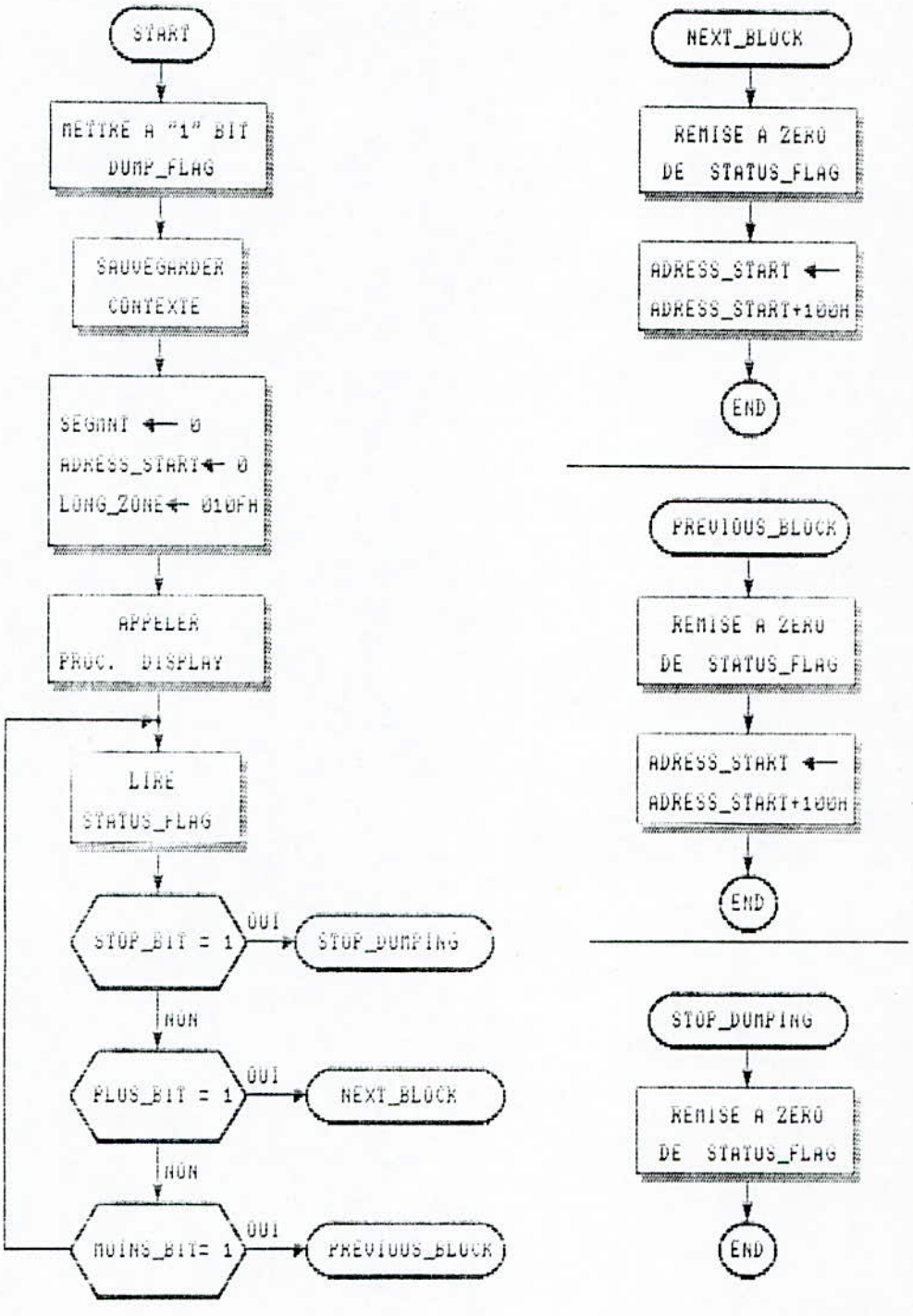
procedure send_character



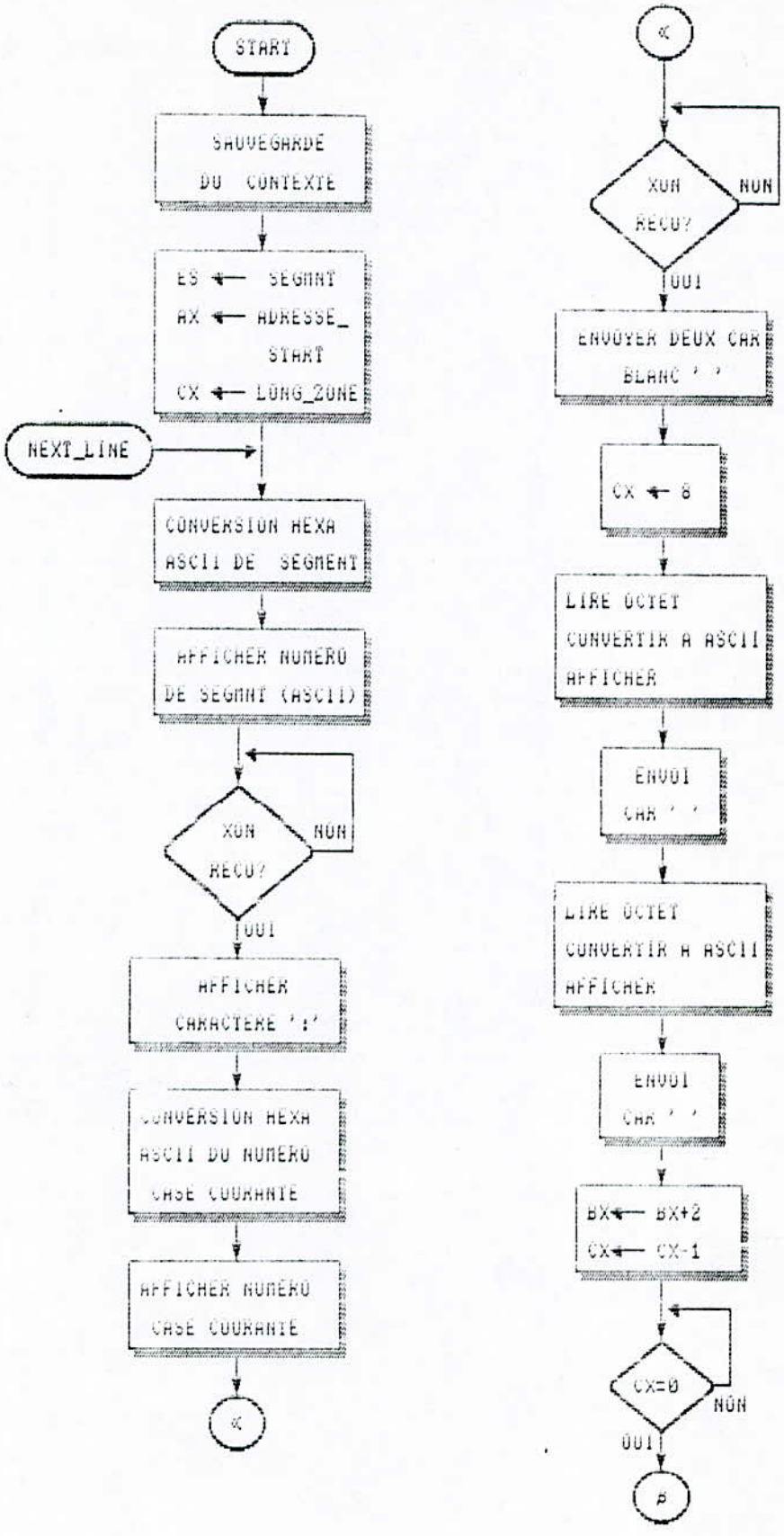
procedure read_character

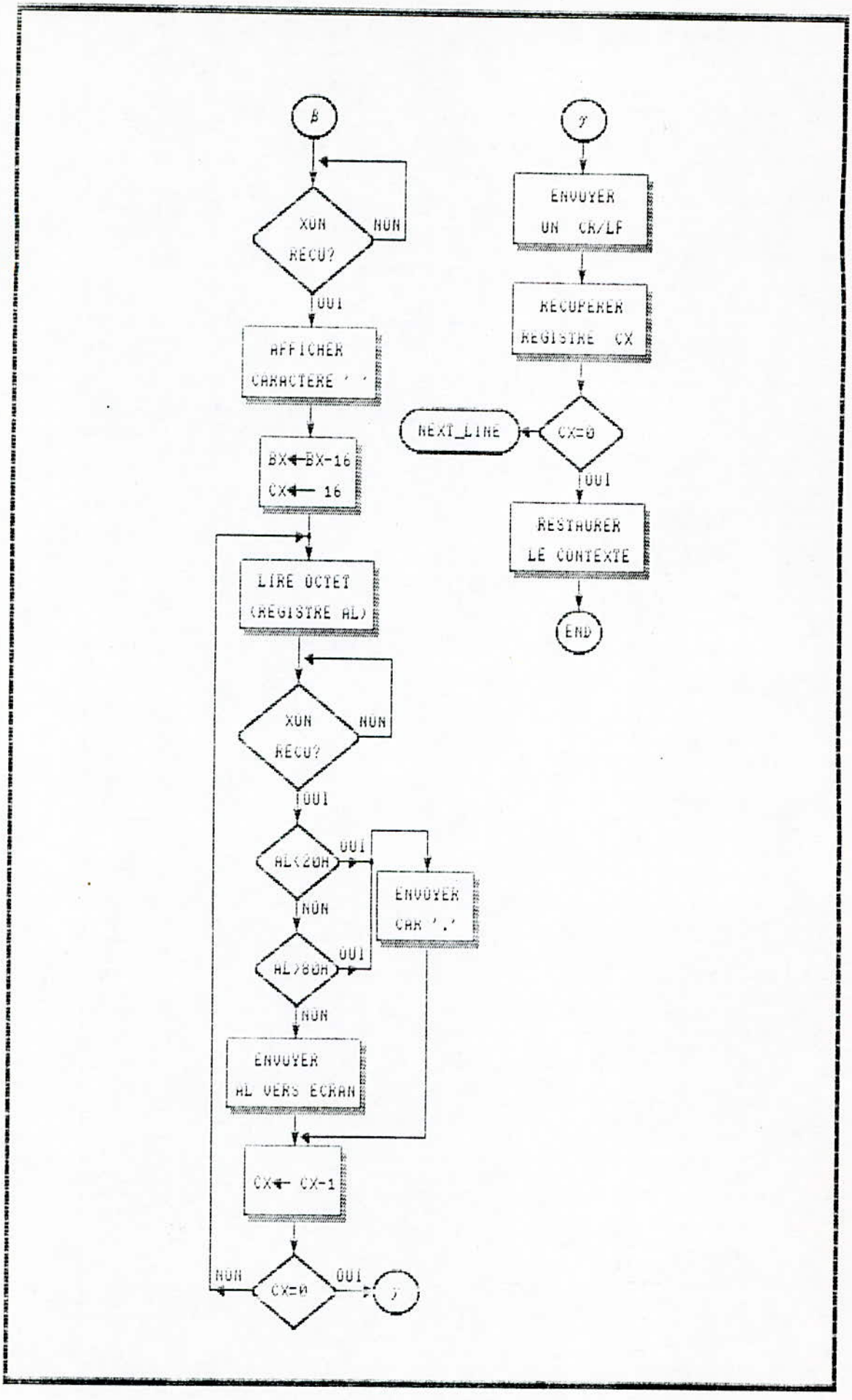


Procedure DUMPALL

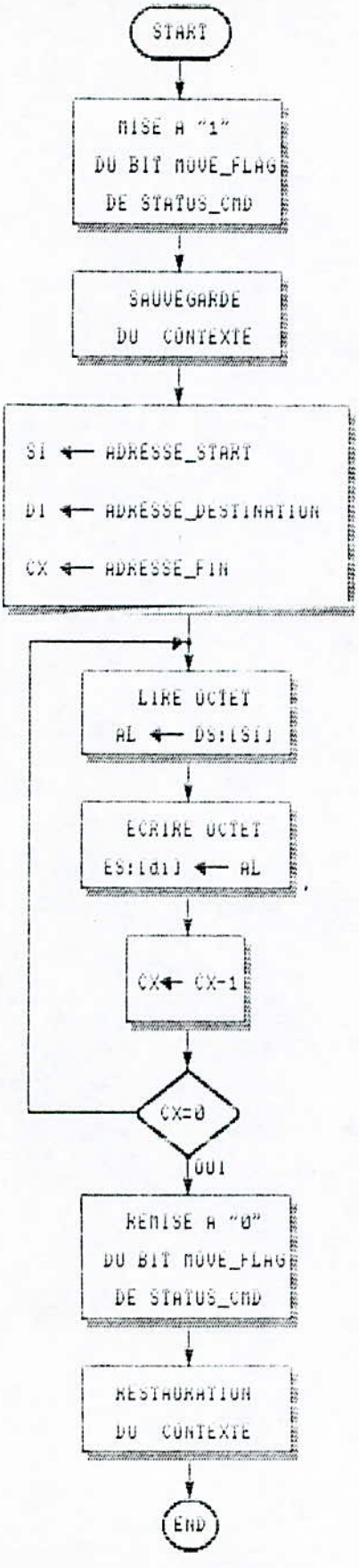


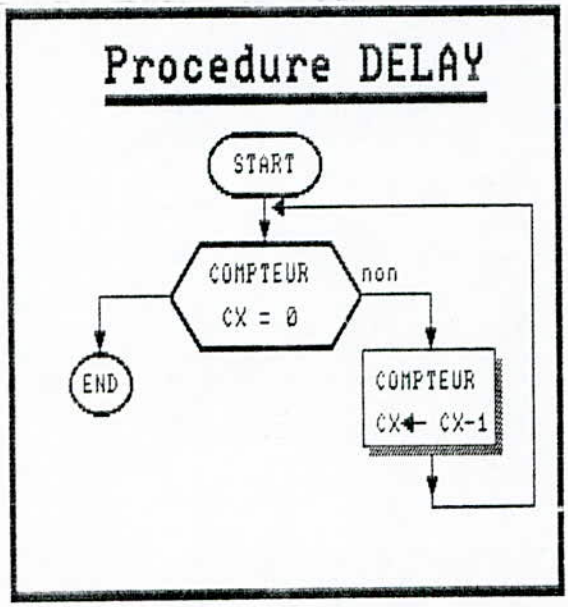
Procedure DISPLAY

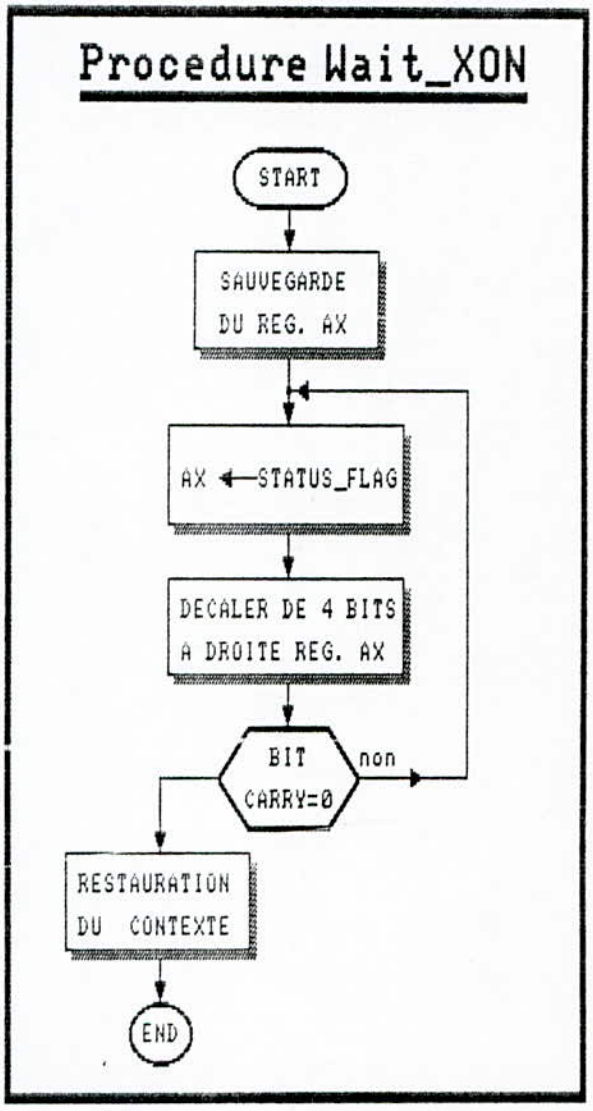


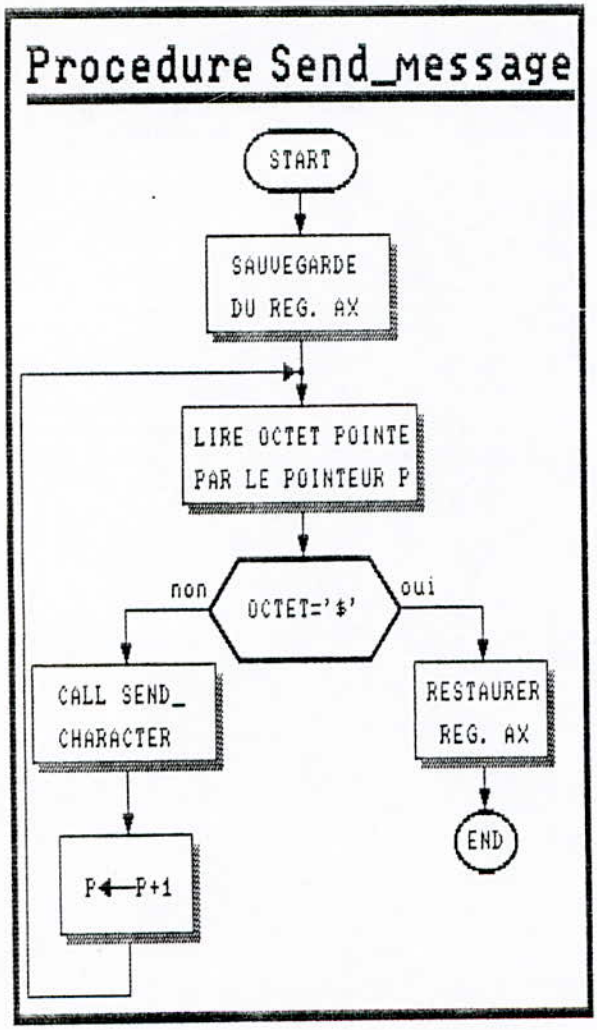

B
XON REÇU?
NON
OUI
AFFICHER CARACTERE ' '
BX←BX-16
CX← 16
LIRE OCTET (REGISTRE AL)
XON REÇU?
NON
OUI
AL<20H
OUI
ENVOYER CAR '.'
NON
AL>80H
OUI
NON
ENVOYER AL VERS ECRAN
CX← CX-1
NON
CX=0
OUI
J
J
ENVOYER UN CR/LF
RECUPERER REGISTRE CX
CX=0
NEXT_LINE
OUI
RESTAURER LE CONTEXTE
END

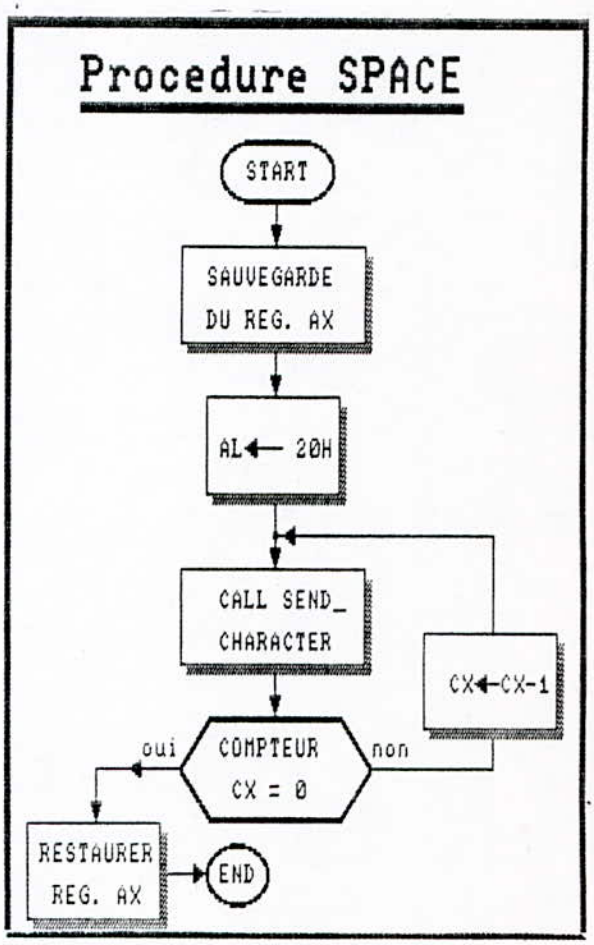
Procedure MOVE

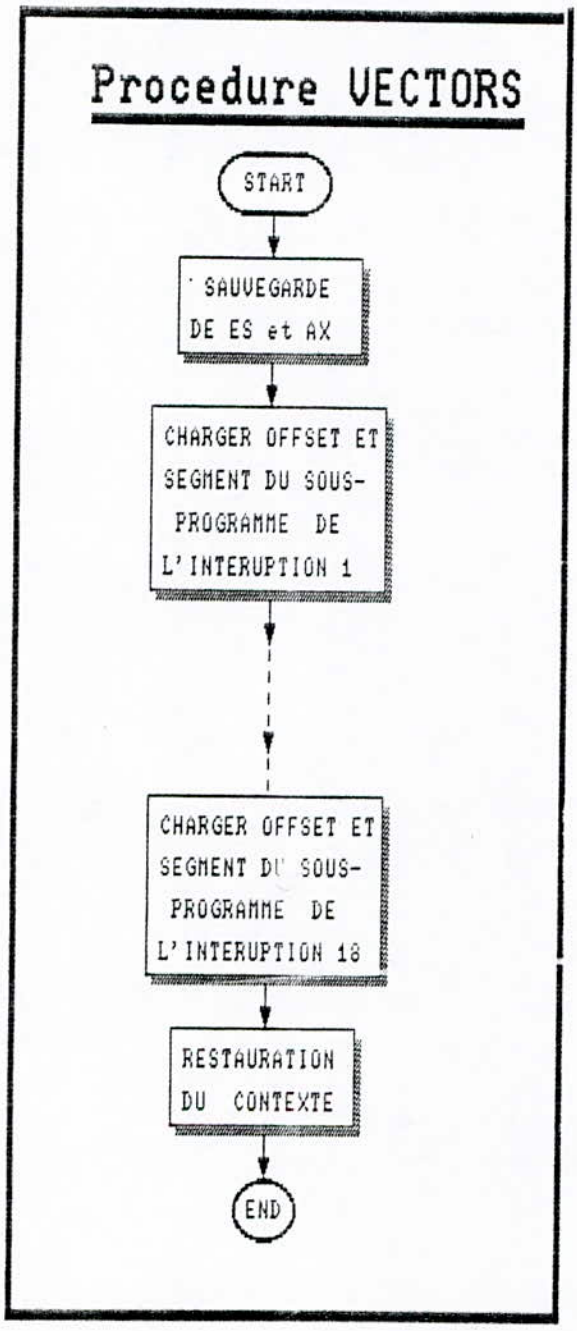


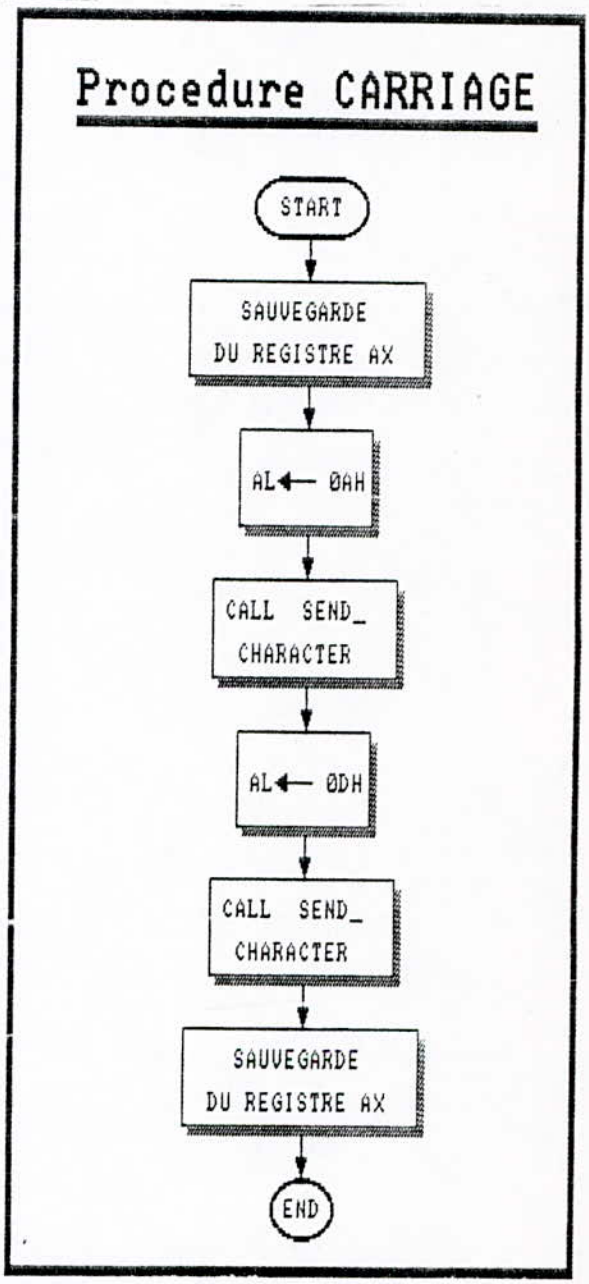

Procedure DELAY
START
COMPTEUR
CX = 0
non
END
COMPTEUR
CX← CX-1

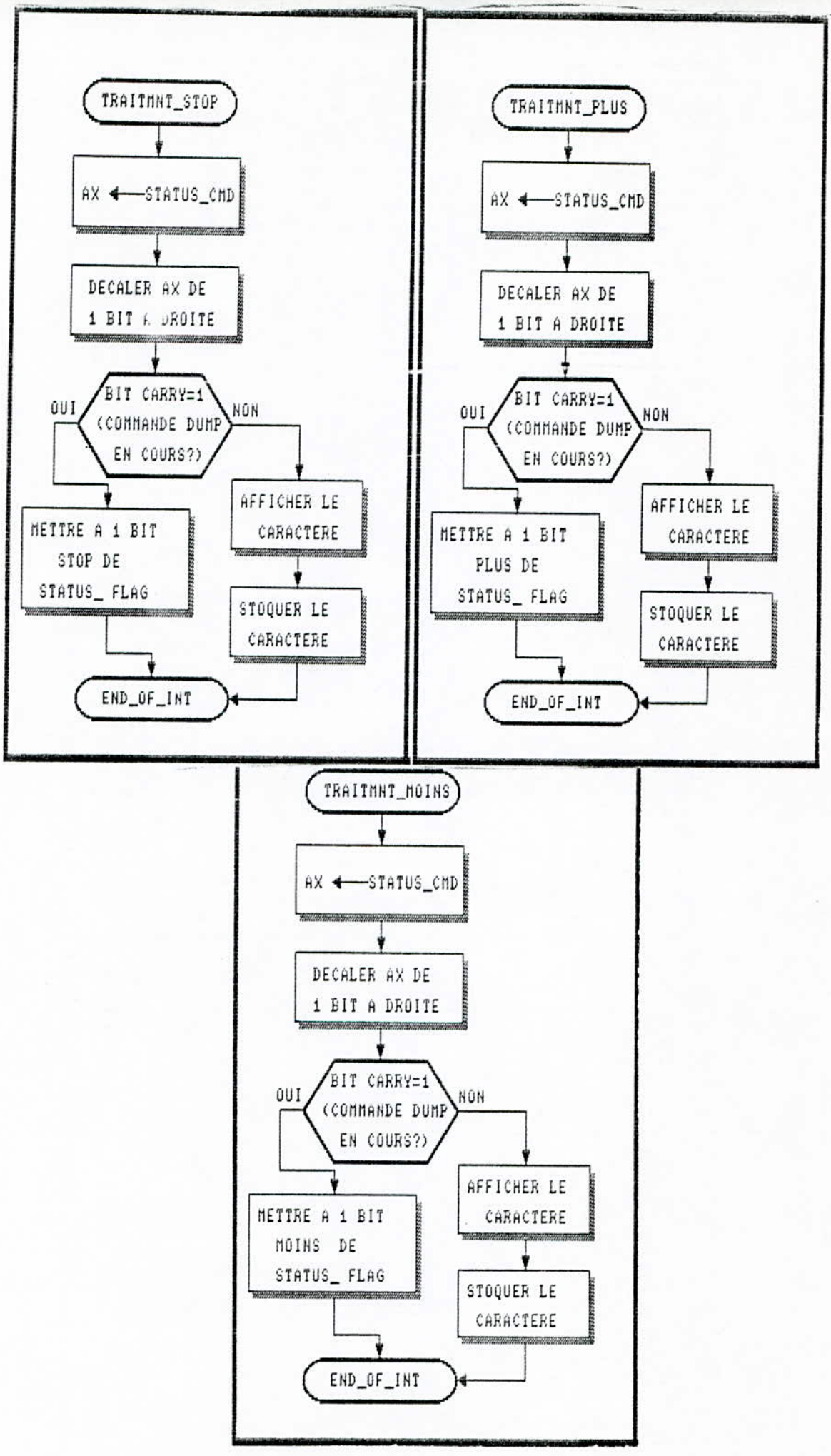

Procedure Wait_XON
START
SAUVEGARDE
DU REG. AX
AX ←STATUS_FLAG
DECALER DE 4 BITS
A DROITE REG. AX
BIT
CARRY=0
non
RESTAURATION
DU CONTEXTE
END

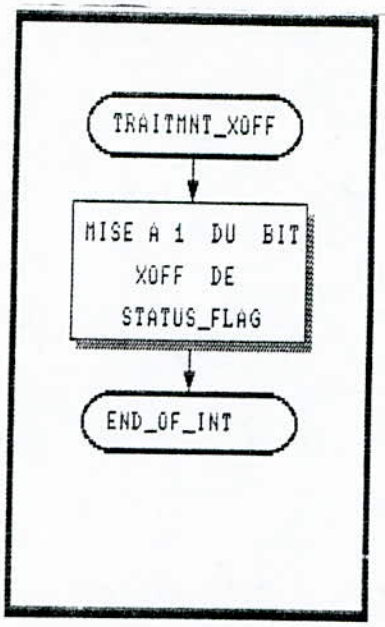

Procedure Send_message
START
SAUVEGARDE DU REG. AX
LIRE OCTET POINTE PAR LE POINTEUR P
non
OCTET='$'
oui
CALL SEND_ CHARACTER
RESTAURER REG. AX
P←P+1
END

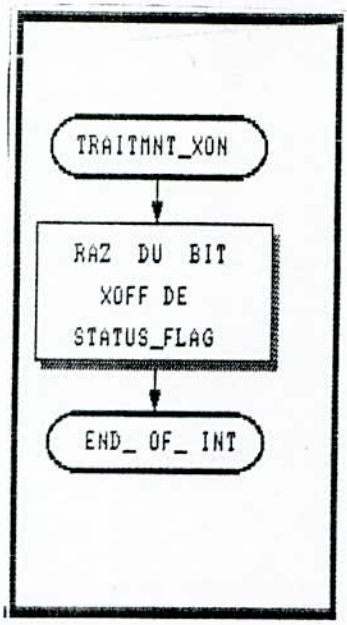

Procedure SPACE
START
SAUVEGARDE DU REG. AX
AL← 20H
CALL SEND_ CHARACTER
CX←CX-1
oui
COMPTEUR CX = 0
non
RESTAURER REG. AX
END

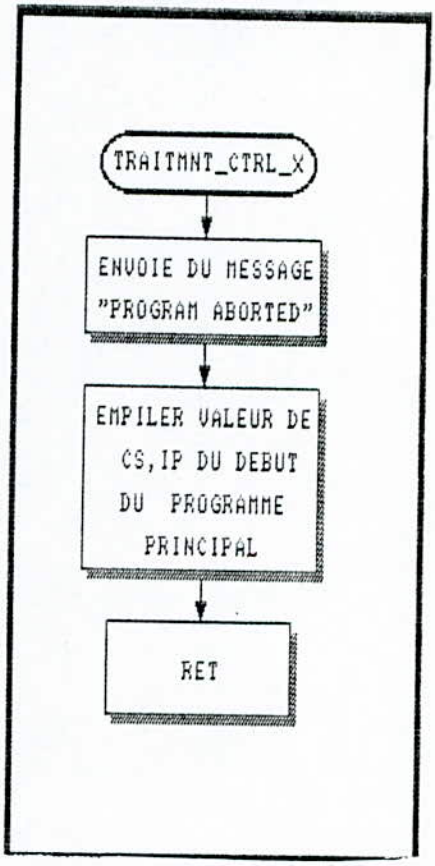

Procedure VECTORS
START
SAUVEGARDE DE ES et AX
CHARGER OFFSET ET SEGMENT DU SOUS-PROGRAMME DE L'INTERUPTION 1
CHARGER OFFSET ET SEGMENT DU SOUS-PROGRAMME DE L'INTERUPTION 18
RESTAURATION DU CONTEXTE
END


Procedure CARRIAGE
START
SAUVEGARDE DU REGISTRE AX
AL ← 0AH
CALL SEND_ CHARACTER
AL ← 0DH
CALL SEND_ CHARACTER
SAUVEGARDE DU REGISTRE AX
END


TRAITMNT_STOP
AX ←STATUS_CMD
DECALER AX DE
1 BIT A DROITE
BIT CARRY=1
(COMMANDE DUMP
EN COURS?)
OUI
NON
METTRE A 1 BIT
STOP DE
STATUS_ FLAG
AFFICHER LE
CARACTERE
STOQUER LE
CARACTERE
END_OF_INT
TRAITMNT_PLUS
AX ←STATUS_CMD
DECALER AX DE
1 BIT A DROITE
BIT CARRY=1
(COMMANDE DUMP
EN COURS?)
OUI
NON
METTRE A 1 BIT
PLUS DE
STATUS_ FLAG
AFFICHER LE
CARACTERE
STOQUER LE
CARACTERE
END_OF_INT
TRAITMNT_MOINS
AX ←STATUS_CMD
DECALER AX DE
1 BIT A DROITE
BIT CARRY=1
(COMMANDE DUMP
EN COURS?)
OUI
NON
METTRE A 1 BIT
MOINS DE
STATUS_ FLAG
AFFICHER LE
CARACTERE
STOQUER LE
CARACTERE
END_OF_INT


TRAITMNT_XOFF
MISE A 1 DU BIT XOFF DE STATUS_FLAG
END_OF_INT


TRAITMNT_XON
RAZ DU BIT XOFF DE STATUS_FLAG
END_ OF_ INT


TRAITMNT_CTRL_X
ENVOIE DU MESSAGE "PROGRAM ABORTED"
EMPILER VALEUR DE CS,IP DU DEBUT DU PROGRAMME PRINCIPAL
RET

Annexe J : Programmes

```
;*****************************************************************************
;                        ROUTINE D'INTIALISATION.
;*****************************************************************************
code            segment
                assume cs:code,ds:data,ss:stack
                ORG     1C00H
                JMP     START           ; START OF PROGRAM
;*****************************************************************************
;                        MESSAGES , VARIABLES , CONSTANTES
;-----------------------------------------------------------------------------
start_message DB 10,13,10,13,10,13,10,13,10,13,10,13
              DB 10,13,'                i80186  IN-CIRCUIT EMULATOR',10,13
              DB 10,13,'                      MONITOR PROGRAM      ',10,13
              DB 10,13,10,13,10,13,10,13,10,13,10,13,'$'
;-----------------------------------------------------------------------------
;                                EQUATES
;-----------------------------------------------------------------------------
        adr_clavier     equ     0
        adr_ecran       equ     0
        ;*************  mapping de la mémoire **********************
        umcs_reg        equ     0FFA0H
        lmcs_reg        equ     0FFA2H
        pacs_reg        equ     0FFA4H
        mpcs_reg        equ     0FFA8H
        mmcs_reg        equ     0FFA6H
        relo_reg        equ     0FFFEH
        umcs_val        equ    0FE38H  ;0 Ws ROM  8KBytes mem haute
        lmcs_val        equ     0FF8H  ;0 Ws RAM  64KBytes mem basse
        pacs_val        equ     003BH  ;2ws ,  PBA = 0
        mpcs_val        equ     8238H  ;0 Ws RAM 16KBytes mem intermediaire
        mmcs_val        equ     09F8H
        relo_val        equ     20FFH
        ;*****************    initialisation de DMA   ****************
        dma_ctl_0       equ    0FFCAH
        dma_ctl_1       equ    0FFDAH
        dma_val_0       equ     0000H
        dma_val_1       equ     0000H
        ;*****************          initialisation de timer  *********
        timer_0_ctr     equ     0FF56H
        timer_1_ctr     equ     0FF5EH
        timer_2_ctr     equ     0FF66H
        ;******* initialisation du controleur d'interuptions ********
        int0_control    equ     0FF38H
        int1_control    equ     0FF3AH
        int2_control    equ     0FF3CH
        int3_control    equ     0FF3EH
        is_reg          equ     0FF2CH
        int_mask        equ     0FF28H
        pr_mask         equ     0FF2AH
        int_status      equ     0FF30H
        int_request     equ     0FF2EH
        poll_status     equ     0FF26H
        poll_reg        equ     0FF24H
        eoi_reg         equ     0FF22H
        init_dma_1      equ     0FF36H
        init_dma_0      equ     0FF34H
        init_timer      equ     0FF32H
        ;-------------------------------
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%                                                                       %%%
```

```
;%%%            CORPS DE LA ROUTINE D'INITIALISATION                    %%%
;%%%                                                                    %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
START:
                cli     ;Inhiber toutes les int.
                assume cs:code,ds:data,ss:stack,es:code
                mov     ax,data
                mov     ds,ax
                mov     ax,cs
                mov     es,ax
                mov     ax,stack
                mov     ss,ax
                lea     sp,stack_top
                ; Fin d'initialisation des registres  DS,SS et SP.
;*********************************************************************
;*                    Routine d'initialisation
;*********************************************************************
; Mise en garde:
;Cette routine d'initialisation ne peut etre mise sous forme de procedure,...
;...du moment que toute procedure met en jeu la pile,qui n'est pas encore...
; ...validée (avant l'initialisation).
;
initialisation:
        ;*******************   MAPPING DE LA MEMOIRE  *************
                mov     dx,umcs_reg
                mov     ax,umcs_val
                out     dx,ax
                mov     dx,lmcs_reg
                mov     ax,lmcs_val
                out     dx,ax
                mov     dx,pacs_reg
                mov     ax,pacs_val
                out     dx,ax
                mov     dx,mpcs_reg
                mov     ax,mpcs_val
                out     dx,ax
                mov     dx,mmcs_reg
                mov     ax,mmcs_val
                out     dx,ax
;Envoi du message 'Ok' sans faire appel à la procedure d'affichage...
;...de messages.
;envoi direct du message pour etre  independant de la pile (cas de...
;...RAM defectueuse).
                mov     ax,'kO'
                mov     dx,adr_ecran
                out     dx,ax
                xchg    ah,al
                out     dx,ax
                ;**********      initialisation     DMA  ************
                mov     dx,dma_ctl_0
                mov     ax,dma_val_0
                out     dx,ax
                mov     dx,dma_ctl_1
                mov     ax,dma_val_1
                out     dx,ax
                ;**********       initialisation    timer  ********
                mov     ax,0000H
                mov     dx,timer_0_ctr
                out     dx,ax
                mov     dx,timer_1_ctr
```

```
                out     dx,ax
                mov     dx,timer_2_ctr
                out     dx,ax
                ;***  initialisation du contrôleur d'interruption  ***
                mov     ax,0001H
                mov     dx,int0_control
                out     dx,ax
                mov     ax,0004H
                mov     dx,int1_control
                out     dx,ax
                mov     ax,0005H
                mov     dx,int2_control
                out     dx,ax
                mov     ax,0006H
                mov     dx,int3_control
                out     dx,ax
                xor     ax,ax
                mov     dx,is_reg
                out     dx,ax
                mov     ax,0007H
                mov     dx,pr_mask
                out     dx,ax
                mov     ax,0000H
                mov     dx,int_request
                out     dx,ax
                mov     ax,0
                mov     dx,int_mask
                out     dx,ax
                mov     ax,0000H
                mov     dx,int_status
                out     dx,ax
                mov     ax,0000H
                mov     dx,poll_status
                out     dx,ax
                mov     ax,0000H
                mov     dx,poll_reg
                out     dx,ax
                mov     ax,8000H
                mov     dx,eoi_reg
                out     dx,ax
                mov     ax,0003H
                mov     dx,init_dma_1
                out     dx,ax
                mov     ax,0002H
                mov     dx,init_dma_0
                out     dx,ax
                mov     ax,0000H
                mov     dx,init_timer
                out     dx,ax
;*************************************************************************
                JMP     JOB
```

```
;*********************************************************************
;    RECOGNIZE_CMD:RECONNAISSANCE COMMANDE RECUE ET  EXECUTION.
;*********************************************************************
; UTILISATION: CALL RECOGNIZE_CMD
RECOGNIZE_CMD   PROC    NEAR
        mov     si,offset cmd_table  ;table des commandes du moniteur.
        mov     di,ptr_read   ;pointeur de début commande reçue.
        mov     cx,12H  ;nombre de commandes.
        mov     dx,4    ;longueur du nom de la commande.
        mov     bx,0
next_cmd:
        mov     ax,es:[bx+si]
        cmp     ax,ds:[bx+di]
        jz      next_car1
next:   add     si,4    ;next command
        MOV     DX,4
        mov     bx,0
        loop    next_cmd
        jmp     unknown_cmd
next_car1:
        inc     bx
        inc     bx
        dec     dx
        dec     dx
        jnz     next_cmd
;Calcul de l'adresse de la routine à appeler...
;... Si CX=4 alors appeler la routine 1 (INIT)
;...     CX=3    //      //      //  //      2 (INIT)
;...     CX=2    //      //      //  //      3 (DUMP)
;...     CX=1    //      //      //  //      4 (DUMP)
;L'adresse de chaque routine est calculée suivant la valeur de CX:
;    ADRESSE : ADR_CMD_TABLE+2(CX-1)
        mov     bx,offset adr_cmd_table
        sub     cx,1    ;cx=cx-1
        shl     cx,1    ;cx=2*cx
        add     bx,cx
        CALL    ES:[BX]
        JMP     end_of_recognize
unknown_cmd:
        lea     si,error_cmd
        call    write_message
end_of_recognize:
        mov     ax,ptr_write    ;Egaliser le pointeur de lecture...
        mov     ptr_read,ax     ;         ...au pointeur d'écriture.
        RET
RECOGNIZE_CMD   endp
;*********************************************************************
```

```
;;*****************************************************************************
;                                   STORAGE
;*****************************************************************************
STORE  PROC    NEAR
    mov     bx,ptr_read  ;tester si dernier car reçu est CR,cela...
    cmp     bx,ptr_write ;...revient à tester si ptr_write=ptr_read.
    jnz     storage
    mov     bx,offset end_of_fifo
    sub     bx,35 ; 35 :longueur maximale nécéssaire pour 1 commande.
    cmp     bx,ptr_write ;s'assurer qu'il y a assez de place pour...
    jge     storage      ;...une nouvelle commande.
    mov     bx,offset start_of_fifo
    mov     ptr_write,bx ;reinitialiser le pointeur au début de fifo.
    mov     ptr_read,bx
storage:
    mov     bx,ptr_write
    mov     byte ptr ds:[BX],al
    add     bx,1    ;next byte.
    mov     ptr_write,bx
    ret
STORE  ENDP
;*****************************************************************************
```

```
;*****************************************************************************
;                              INTERRUPTION   CLAVIER
;*****************************************************************************
;SOUS_PROGRAMME_INT0
;NB: Si le  traitant d'int  s'avere  très  long  il  suffit  de declarer...
;...chaque traitement en procedure.
;.................................
;L'interuption peut paraitre à n'importe quel instant,plus particulièrement...
;...lors de l'éxécution d'1 commande.Le contexte général(entre autre ES,DS,...
;...CS et SS)peut changer localement dans cette commande d'où la necèssité...
;...de le recuperer au début du traitant d'interuption et de sauvegarder le...
;...contexte de la commande interrompue.
                org     1A00H
                CLI
DB      60H     ;pusha
; Sauvegarde du contexte de la commande interrompue(CS et IP sauvegardés ...
;...automatiquement).
push    es
push    ds
pushf
;Récupération du contexte général du programme (CS,DS,ES,SS).
                assume  cs:code,ds:data,es:code,ss:stack
                mov     ax,data
                mov     ds,ax
                mov     ax,cs
                mov     es,ax
;.....................................................................
                ;lire un caractere.
                XOR     AH,AH
                CALL    READ_CHAR
                PUSH    AX
                ;test du caractere.
                CMP     AL,18H  ;code de ctrl_x
                JZ      ctrl_x_traitemnt
                cmp     al,11H          ;XON
                jz      XON_traitemnt
                cmp     al,13H          ;XOFF
                jz      XOFF_traitemnt
                cmp     al,'s'
                jz      Stop_traitemnt
                cmp     al,'S'
                jz      stop_traitemnt
                cmp     al,2BH          ;code de "+".
                jz      plus_traitemnt
                cmp     al,2DH
                jz      moins_traitemnt
                ;Si l'une des commandes suivantes(dump,move,areg,...)est activ
                ;...le caractere reçu ne devra pas etre envoyé à l'écran.
                mov     ax,status_cmd
                sar     ax,1
                jc      no_valid_cmd1
                ;.............................
                pop     ax
                push    ax
                cmp     al,1AH          ;Ctrl Z
```

```
XOFF_traitement:
                mov     ax,xoff_mask
                or      status_flag,ax ;XOFF reçu,mettre'1'dans le flag XOFF.
                pop     ax
                jmp     end_of_int1
XON_traitement:
                mov     ax,not xoff_mask
                and     status_flag,ax ;XON reçu,mettre '0' dans le flag XOFF.
                pop     ax
                jmp     end_of_int1
ctrl_x_traitement:
                ;"Ctrl X"(Correspondant à l'ABORT),...
                ;...abandonner le programme et rectifier les registres...
                ;...sauvegardés qui sont les flags,CS et IP.
                ;...Aprés l'abort,revenir au mode commande.
                lea     si,abort
                call    write_message
                lea     sp,stack_top
                ;revoir cette valeur.
                mov     ax,0F202H       ;Valeur de SW par défaut avec IF=1.
                push    ax
                mov     ax,seg wait_for_cmd1  ;Nouvelle valeur de CS.
                push    ax
                mov     ax,offset wait_for_cmd1  ;Nouvelle valeur de IP.
                push    ax
                mov     ax,8000H        ; Envoi du EOI...
                mov     dx,eoi_reg      ; ...dû au fait que c'est une...
                out     dx,ax           ; ...int materielle(mode non specifiqu
                IRET
BS_traitement:
                ;Est-ce un début de commande?si oui ne pas envoyer le BS...
                ;...sinon le prompt sera alteré.
                ;...cela revient à tester si PTR_WRITE est égal à PTR_READ.
                mov     ax,ptr_write
                cmp     ptr_read,ax
                pop     ax
                jz      end_of_int1
                CALL    DELETE_CHAR
                ;decrementer ptr_write.
                mov     bx,ptr_write
                dec     bx
                mov     ptr_write,bx
                jmp     end_of_int1
LF_traitement:
                call    LF_PROC
                pop     ax
                call    store
                jmp     end_of_int1
CR_traitement:
                CALL    CR_PROC
                pop     ax
                call    store
                jmp     end_of_int1
Ctrl_Z_traitement:
                ;Si Ctrl_z reçu et si la commande en cours est la SREG  ...
```

```
                jz      ctrl_Z_traitemnt
                cmp     al,08H          ;Back Space
                jz      BS_traitemnt
                cmp     al,0AH          ;line feed
                jz      LF_traitemnt
                cmp     al,0DH          ;carriage return
                jz      CR_traitemnt
                ;Les car qui ne doivent pas etre réenvoyés (XON,XOFF,CtrlX..).
                ;...devront etre testés avant l'instruction d'affichage suiva
                pop     ax
display_store:
                call    affich
                call    store
                jmp     end_of_intl
;Comme les sauts conditionnels ne permettent pas des deplacement superieur...
;...à 129 octets , on a opté pour un double saut .
stop_traitemnt:         jmp     stop_traitement
NO_VALID_CMD1:          JMP     NO_VALID_CMD
ctrl_Z_traitemnt:       jmp     ctrl_z_traitement
ctrl_x_traitemnt:       jmp     ctrl_x_traitement
xon_traitemnt:          jmp     XON_traitement
xoff_traitemnt:         jmp     XOff_traitement
plus_traitemnt:         jmp     plus_traitement
moins_traitemnt:        jmp     moins_traitement
BS_traitemnt:           jmp     bs_traitement
lf_traitemnt:           jmp     lf_traitement
cr_traitemnt:           jmp     cr_traitement
;*******************************  traitements  ********************************
stop_traitement:mov     ax,status_cmd
                sar     ax,1
                pop     ax
                jnc     DISPLAY_STORE ;Si commande DUMP non valide,...
                mov     ax,stop_mask    ;...afficher et stoquer le caractere.
                or      status_flag,ax
                jmp     end_of_intl
plus_traitement:
                ;Tester si la commande DUMP est activée,si oui alors ...
                ;...positionner flag PLUS_MASK sinon l'afficher et le stoquer.
                mov     ax,status_cmd
                sar     ax,1
                pop     ax
                jnc     display_store
                mov     ax,plus_mask
                or      status_flag,ax
                jmp     end_of_intl
moins_traitement:
                ;Tester si la commande DUMP est active ,si oui alors ...
                ;...positionner flag MOINS_MASK sinon l'afficher.
                mov     ax,status_cmd
                sar     ax,1
                pop     ax
                jnc     display_store
                mov     ax,moins_mask
                or      status_flag,ax
                jmp     end_of_intl
```

```
                ;...ou la GREG (commandes de modification des regsitres)...
                ;...alors valider les valeurs des registres(positionner...
                ;...ctrl_Z_flag).
                ;Tester si l'une des commandes SREG ou GREG est en cours.
                mov     ax,status_cmd
                mov     cl,3
                sar     ax,cl
                jc      valid_cmd       ;Commande SREG en cours.
                sar     ax,1
                jnc     no_valid_cmd    ;Aucune des deux commandes...
                                        ;...(SREG et   GREG)n'est validées.
valid_cmd:      mov     ax,ctrl_Z_mask
                or      status_flag,ax
no_valid_cmd:   pop     ax
                jmp     end_of_int1
end_of_int1:
popf
pop     ds
pop     es
DB      61H     ;popa
end_of_int2:
; RESTAURATION du contexte de la commande interrompue(CS et IP restaurés
;...automatiquement).
                mov     ax,8000H        ; Envoi du EOI...
                mov     dx,eoi_reg      ; ...dû au fait que c'est une...
                out     dx,ax           ; ...int materielle(mode non specific)
        iret
;SOUS_PROGRAMME_INT0    .  FIN
;*****************************************************************************
```

BIBLIOGRAPHIE:

* "INTRODUCTION TO THE 80186" MICROSYTEMS COMPONENTS. INTEL 1989

* "Methodes de développement d'un système à microprocesseur".
A.AMGHAR. 1989. EDITION

* "Programmation en assembleur 8086/8088 ".
F.RETAUREAU. EDITION SYBEX.

* "Programmation en langage assembleur 8088/8086
iAPX 186,188,286 et 386". B.GEOFFRION. EDITION RADIO.

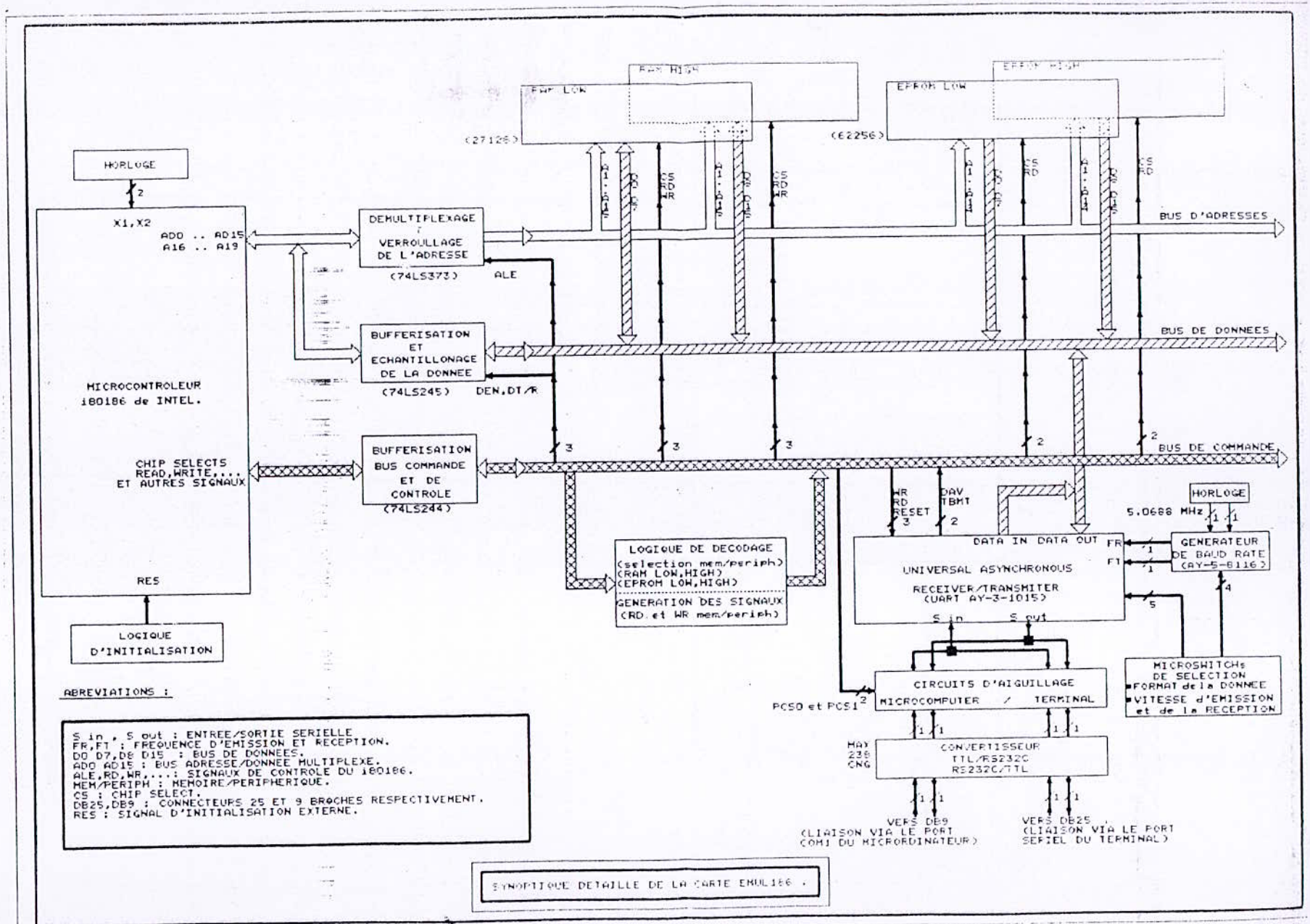
* "Conception et réalisation d'une carte unité centrale à base du microcontrôleur i80186 de INTEL."
Projet de fin d'études "Ecole Nle Polytechnique."JUIN 1990
D.KHADRAOUI

* "Choisir un système de développement pour microprocesseur".
M.BLANCHARD,J.C.CAVARROC,M.GAY EDITION éditESTS.

* "Systèmes d'exploitation des micro-ordinateurs."
M. Courvoisier
R. Valette edition Dunod 1986

* "Systèmes d'exploitation des micro-ordinateurs."
M. Dahmkee Mc.Graw-Hill 1984

* "Fonction d'émulation assistée par IBM-PC"
15 septembre 1984.



ABREVIATIONS :

S in , S out : ENTREE/SORTIE SERIELLE.
FR,FT : FREQUENCE D'EMISSION ET RECEPTION.
D0 D7,D8 D15 : BUS DE DONNEES.
AD0 AD15 : BUS ADRESSE/DONNEE MULTIPLEXE.
ALE,RD,WR,...: SIGNAUX DE CONTROLE DU i80186.
MEM/PERIPH : MEMOIRE/PERIPHERIQUE.
CS : CHIP SELECT.
DB25,DB9 : CONNECTEURS 25 ET 9 BROCHES RESPECTIVEMENT.
RES : SIGNAL D'INITIALISATION EXTERNE.

SYNOPTIQUE DETAILLE DE LA CARTE EMUL186 .

page 1

page 2:

maintenance étape -

61 - moniteur.

X Question ALS → RW)

USART

▷ Comment vérifier le flux → (Debit?)

intérêt pas de perte d'information.

est-ce une question de formatage de donnée.

- Emulation / simulation

Tout dans un micro-