Test comportemental de microprocesseurs

Raoul Velazco

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
Génie Informatique

par

Raúl VELAZCO

TEST COMPORTEMENTAL DE MICROPROCESSEURS

Thèse soutenue le 18 mars 1982 devant la Commission d'Examen :

Président	: Monsieur	G. VEILLON
Examinateurs	: Messieurs	A. COSTES
		F. GRILLOT
		C. OUANNES
	Mademoiselle	Ch. ROBACH
	Madame	G. SAUCIER
	Monsieur	D. SIEWIOREK

INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE

Année universitaire 1981-1982

Président : Daniel BLOCH

Vice-Présidents : René CARRÉ
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES

ANCEAU François	E.N.S.I.M.A.G
BARRAUD Alain	E.N.S.I.E.G
BESSON Jean	E.N.S.E.E.G
BLIMAN Samuel	E.N.S.E.R.G
BLOCH Daniel	E.N.S.I.E.G
BOIS Philippe	E.N.S.H.G
BONNETAIN Lucien	E.N.S.E.E.G
BONNIER Etienne	E.N.S.E.E.G
BOUVARD Maurice	E.N.S.H.G
BRISSONNEAU Pierre	E.N.S.I.E.G
BUYLE-BODIN Maurice	E.N.S.E.R.G
CAVAIGNAC Jean-François	E.N.S.I.E.G
CHARTIER Germain	E.N.S.I.E.G
CHENEVIER Pierre	E.N.S.E.R.G
CHERADAME Hervé	M.C.P.P
CHERUY Arlette	E.N.S.I.E.G
CHIAVERINA Jean	M.C.P.P
COHEN Joseph	E.N.S.E.R.G
COUMES André	E.N.S.E.R.G
DURAND Francis	E.N.S.E.E.G
DURAND Jean-Louis	E.N.S.I.E.G
FELICI Noël	E.N.S.I.E.G
FOULARD Claude	E.N.S.I.E.G
GENTIL Pierre	E.N.S.E.R.G
GUERIN Bernard	E.N.S.E.R.G
GUYOT Pierre	E.N.S.E.E.G
IVANES Marcel	E.N.S.I.E.G
JAUSSAUD Pierre	E.N.S.I.E.G
JOUBERT Jean-Claude	E.N.S.I.E.G
JOURDAIN Geneviève	E.N.S.I.E.G
LACOUME Jean-Louis	E.N.S.I.E.G
LATOMBE Jean-Claude	E.N.S.I.M.A.G
LEROY Philippe	E.N.S.H.G
LESIEUR Marcel	E.N.S.H.G
LESPINARD Georges	E.N.S.H.G
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G
MAZARE Guy	E.N.S.I.M.A.G
MOREAU René	E.N.S.H.G
MORET Roger	E.N.S.I.E.G
MOSSIERE Jacques	E.N.S.I.M.A.G
PARIAUD Jean-Charles	E.N.S.E.E.G
PAUTHENET René	E.N.S.I.E.G
PERRET René	E.N.S.I.E.G
PERRET Robert	E.N.S.I.E.G

PIAU Jean-Michel	E.N.S.H.G
POLOUJADOFF Michel	E.N.S.I.E.G
POUPOT Christian	E.N.S.E.R.G
RAMEAU Jean-Jacques	E.N.S.E.E.G
RENAUD Maurice	M.C.P.P
ROBERT André	M.C.P.P
ROBERT François	E.N.S.I.M.A.G
SABONNADIERE Jean-Claude	E.N.S.I.E.G
SAUCIER Gabrielle	E.N.S.I.M.A.G
SCHLENKER Claire	E.N.S.I.E.G
SCHLENKER Michel	E.N.S.I.E.G
SERMET Pierre	E.N.S.E.R.G
SOUQUET Jean-Louis	E.N.S.E.E.G
SILVY Jacques	M.C.P.P
SOHM Jean-Claude	E.N.S.E.E.G
VEILLON Gérard	E.N.S.I.M.A.G
ZADWORNY François	E.N.S.E.R.G

PROFESSEURS ASSOCIES

GANDINI Alessandro	M.C.P.P
MAXWORTHY Thony	E.N.S.H.G
MROVEC Stanislas	E.N.S.E.E.G
PARRIAUX Olivier	E.N.S.I.E.G
PEISNER Janos	E.N.S.E.R.G

PROFESSEURS E.N.S MINES SAINT ETIENNE

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S (Directeurs et Maîtres de recherche)

FRUCHART Robert	Directeur de recherche
ALLIBERT Michel	Maître de recherche
ANSARA Ibrahim	Maître de recherche
CARRE René	Maître de recherche
DAVID René	Maître de recherche
DRIOLE Jean	Maître de recherche
KAMARINOS Georges	Maître de recherche
KLEITZ Michel	Maître de recherche
LANDAU Ioan-Doré	Maître de recherche
MERMET Jean	Maître de recherche
MUNIER Jacques	Maître de recherche
VERDILLON André	Maître de recherche

CHERCHEURS DU MINISTERE DE L'INDUSTRIE

(Directeurs et Maîtres de recherche - E.N.S Mines Saint Etienne)

LESBATS Pierre	Directeur de recherche
BISCONDI Michel	Maître de recherche
KOBYLANSKI André	Maître de recherche
LE COZE Jean	Maître de recherche
THEVENOT François	Maître de recherche
TRAN MINH Canh	Maître de recherche
LALAUZE René	Maître de recherche
LANCELOT Francis	Maître de recherche

PERSONNALITES HABILITEES A DIRIGER DES TRAVAUX DE RECHERCHE

(Décision du Conseil Scientifique)

E.N.S.E.E.G

BERNARD Claude
BONNET Roland
CAILLET Marcel
CHATILLON Catherine
COULON Michel
EUSTATHOPOULOS Nicolas
HAMMOU Abdelkader
JOUD Jean-Charles
MALMEJAC Yves (CENG)
RAVAINE Denis
SAINFORT (CENG)
SARRAZIN Pierre
TOUZAIN Philippe
URBAIN Georges (Laboratoire des Ultraréfractaires, ODEILLO)

E.N.S.M Saint Etienne

GUILHOT Bernard
THOMAS Gérard
DRIVER Julian

E.N.S.E.R.G

BOREL Joseph
CHEHIKIAN Alain

E.N.S.I.E.G

BORNARD Guy
DESCHIZEAUX Pierre
GLANGEAUD François
LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G

DELHAYE Jean-Marc

E.N.S.I.M.A.G

A mis padres,

a mis hermanos,

a mis amigos...

... a mi guitarra .

Je tiens à exprimer toute ma reconnaissance à Madame Gabrièle SAUCIER, Professeur à l'ENSIMAG, de m'avoir accueilli dans son équipe de recherche, de m'avoir donné le goût de la recherche et d'avoir dirigé mes travaux.

Je tiens à remercier Monsieur Gérard VEILLON, Professeur à l'ENSIMAG, de m'avoir fait l'honneur de présider le jury de cette thèse,

- Monsieur François GRILLOT, Chef du service test et CAO à E.S.D., d'avoir accepté dêtre rapporteur de cette thèse et d'avoir facilité le dialogue avec l'industrie,

- Mademoiselle Chantal ROBACH, chargée de recherche au CNRS, d'avoir animé les recherches en me faisant toujours part de ses remarques constructives.

Je tiens également à remercier,

- Monsieur A. COSTES, Professeur à l'Université de Toulouse,

- Monsieur C. OANNES, chargé de mission à l'Agence de l'Informatique,

et,

- Monsieur D. STEWIOREK, Professeur à Carnegie-Mellon University,

d'avoir accepté de faire partie de ce jury.

Que soient ici remerciés :

- Tous mes collègues et amis de l'équipe "Conception et Sécurité des Systèmes" de l'ambiance chaleureuse et amicale qu'ils ont su créer pendant toutes ces années passées loin de mon pays et de ma famille.

En particulier, je voudrais remercier C. BELLON d'avoir eu la patience de relire et corriger ces travaux, A LIOTHIN et J.P. EYNARD dont j'ai pu apprécier les qualités lors de notre collaboration.

- Toutes les personnes qui ont assuré la réalisation technique de cet ouvrage : Madame S. ROCHE pour la frappe ainsi que l'équipe de reprographie et Monsieur D. IGLESIAS pour le tirage.

TABLE DES MATIERES

INTRODUCTION

Le problème du test de circuits à très haute intégration est un problème critique non résolu et qui s'aggrave au fur et à mesure des progrès réalisés au niveau de l'intrégation. Il concerne le concepteur de circuits intégrés qui aura à réaliser le test du prototype de son circuit, le fabricant qui sera confronté au problème difficile du tri et de l'assurance qualité du produit et enfin l'usager qui utilisera ces circuits dans les applications qu'il développera.
Ces trois types de problèmes (test en fin de conception, en fin de fabrication, en réception et utilisation) doivent être soigneusement distingués. En effet, ces problèmes se posent de façon très différente et se distinguent par :

. les objectifs recherchés;

. le degré de connaissance du circuit dont on dispose au moment du test; cette caractéristique sépare nettement le concepteur de l'utilisateur de circuits;

. le niveau de description manipulé par le type de test considéré;

. la complexité et le coût (en temps) alloués à l'élaboration des logiciels de test;

. l'équipement de test et la qualification du personnel;

. le temps alloué au test lui-même.

Les méthodes d'élaboration des stratégies et des logiciels de test seront très variables dans ces différentes situations. Le concepteur qui recherche la conformité à ses spécifications doit disposer de ces spécifications et savoir élaborer des tests activant tous les "cas de figure" de son circuit. Les recherches s'orientent vers des méthodes de conception sûre [SAU 81], vers des techniques de validation souvent très proches des techniques de preuve d'algorithmes ou de programme. L'objectif prioritaire n'est pas la détection d'une défaillance de la chaîne de fabrication mais la détection de la non conformité du circuit au cahier de charges initial (contraintes dynamiques entre autres).

On note une évolution d'une part vers des techniques de test de pointe telles que l'analyse dynamique par stroboscopie au microscope à balayage électronique formant des images dynamiques [FOU 81], d'autre part vers des contraintes de testabilité sévères du type LSSD ou"scan-path" [FRA 81]assurant des moyens d'accès de test au circuit et facilitant le travail du concepteur de la stratégie de test.

En fabrication les descriptions structurelles seront utilisées et la génération de vecteurs de test déterministes sur hypothèses de défaillance est la plus souhaitable. Malheureusement, la situation est spécialement "dramatique". En effet, les défaillances sont mal connues et la complexité des circuits actuels rendent impossible leur simulation avec panne et les techniques de génération automatique. Une importante recherche doit s'organiser vers des approches multiniveaux.

	CONCEPTION	EN FABRICATION	POUR USAGERS
Degré de connaissance et niveau de description	Connaissance structurelle tous niveaux Documents de spécifications et de réalisations disponibles		Connaissance "externe" Description comportementale des fonctions usager
Temps d'élaboration des séquences de test	Peut accompagner la conception plusieurs mois/circuit		Réduit car plusieurs types de micro-processeurs
Temps de test	Long (Debugging)	de l'ordre de la seconde/circuit	Variable sec-mn-heure/circuit
Equipement	LOURD : Testeur à pointes Electron beam scanner Testeur Universels		Variable
Buts Détection	Conformité aux spécifications de conception et test du 1er prototype	Défaillance de la chaîne de fabrication	Conformité aux spécifications usagers
Localisation	fine	fine (dans une 2ème étape du test)	localisation non nécessaire

Les tests recherchés doivent être courts (lots importants) et garantir un diagnostic fin (identification de la défaillance) pour assurer un retour sur la chaîne de fabrication.

En ce qui concerne l'usager, il est confronté au problème de la réception des composants et au problème de l'écriture des programmes de test des systèmes (cartes) construits à partir de ces circuits intégrés, aussi bien en fin de conception qu'en fin de fabrication ou en maintenance.

Nous nous sommes ici intéressés au test usager des microprocesseurs. Quelle carte à l'heure actuelle ne comporte pas un microprocesseur et des circuits annexes ? A cours de l'élaboration d'un programme, quels vecteurs de test faudra-t-il envoyer sur ce microprocesseur ?

L'usager ne peut attendre que le fabricant ait résolu le problème difficile du test, c'est-à-dire lui fournisse une description structurelle normalisée, ou mieux encore, des séquences de test homologuées.

De toutes façons, même en supposant qu'une description structurelle lui soit fournie, aucune méthode de test déterministe ne pourrait à l'heure actuelle générer des séquences complètes de test. L'usager doit donc porter ses efforts sur le test du microprocesseur à travers ses fonctions usagers.

Le microprocesseur est un circuit très spécial face au test car ses fonctions usagers qui incluent le jeu d'instructions (instructions avec toutes les possibilités d'adressage) sont en nombre très élevé. Un premier problème qui s'est posé dans ce travail est la description normalisée de ces instructions et la définition d'un test de conformité.

Ce test de conformité est complété par des tests dits test de balayage : envoyer des données de test spécifiques à certaines unités fonctionnelles et par le test de signaux complémentaires (Section I). A la fin de cette section l'objectif principal de la thèse est discutée : peut-on générer de façon quasi-automatique ces programmes de test ? En effet, l'usager de microprocesseurs utilise un grand nombre de circuits différents. Il ne peut en général disposer d'un personnel assurant l'écriture de programmes de test pour chaque nouveau type de microprocesseur utilisé. Les principes de base d'un générateur automatique de programmes de test de microprocesseurs sont donc présentés.

Dans la Section II, une réalisation expérimentale est présentée donnant les résultats d'une première étude de faisabilité. Un certain nombre d'outils logiciels (assembleur paramétrable, langage de test) ont été utilisés ou conçus et ont permis l'élaboration de programmes de test de certains microprocesseurs.

Ces programmes ont été appliqués à des lots de microprocesseurs sur un testeur expérimental conçu et réalisé avec la collaboration du GIS mini-micro, la validité de la méthode étant ainsi prouvée. Les extensions de ce testeur, en vue d'assurer un support définitif de cette méthode de test, sont discutées.

SECTION I

METHODE DE TEST

INTRODUCTION

Dans le paragraphe I on présente l'état de l'art et les orientations significatives dans le domaine du test des microprocesseurs.

Dans le paragraphe II sont discutées les principales caractéristiques de la méthode de test adoptée et de l'environnement dans lequel le test sera mis en oeuvre.

La génération automatique des programmes de test est un objectif visé de ce travail. Pour cela on définit au paragraphe III une modélisation des microprocesseurs. Au paragraphe IV, à l'aide de ce modèle, le problème de l'accès aux éléments de mémorisation internes des microprocesseurs est étudié .

La structure des programmes de test est ensuite présentée (paragraphe V). Les programmes de test de conformité de balayage et des signaux complémentaires pourront être générés automatiquement à partir du modèle proposé. Cette section se termine par l'étude du problème de la génération automatique de ces programmes (paragraphe VI).

I - LES MÉTHODES DE TEST DE MICROPROCESSEURS

Trois types de test sont utilisés pour vérifier les caractéristiques d'un circuit intégré [Fra 81] [Rob 79]

- Des mesures des niveaux de tension et de courant appelés généralement *tests paramétriques statiques*
- Des mesures des paramètres dépendant du temps (temps de montée, de descente, de stockage, de propagation, ...) désignés par *tests paramétriques dynamiques*
- Des tests *logiques ou fonctionnels* permettant de garantir que le circuit réalise sa fonction logique dans des conditions d'environnement semblables à celles de son utilisation (alimentation, température,...).

Les deux premiers tests sont généralement appliqués en fin de fabrication du composant ou en contrôle d'entrée, ces tests comparent les limites réelles des paramètres mesurés à celles spécifiées par le cahier de charges.

Le test logique ou fonctionnel est appliqué en fin de fabrication, en contrôle d'entrée ou en maintenance. Lorsque le circuit est un microprocesseur la définition et la mise en oeuvre d'un tel type de test pose de sérieuses difficultés.

En général les tests logiques comportent les étapes suivantes [Hay 80] :

- La génération des données de test : ce sont les données d'entrée appelées *vecteurs de test* et éventuellement les réponses de sortie.

- L'application des vecteurs de test au circuit à tester.

- L'évaluation des réponses obtenues.

Il existe actuellement plusieurs approches. Elles se différencient essentiellement par la méthode de génération des vecteurs d'entrée et par la mise en oeuvre du test.

A - *METHODES DE GENERATION DES VECTEURS DE TEST*

En ce qui concerne la génération de vecteurs d'entrée on distingue [Rob 79][Rob 80] :

- Les méthodes de test avec *vecteurs prédéterminés* les vecteurs de test sont calculés au cours d'une étude préalable du circuit ; ils sont ensuite appliqués lors du test effectif du circuit.

  L'ensemble des vecteurs de test est en général déterminé pour couvrir au mieux un ensemble de pannes, déterminé d'aprés des hypothèses de pannes.

- Les méthodes *de test aléatoire* : les données de test sont générées aléatoirement lors du test effectif du circuit. Le problème est alors d'estimer la longueur de la séquence de test nécessaire pour assurer une qualité de détection donnée [The 80].

On ne s'interesse ici qu'aux méthodes de test avec vecteurs prédeterminés.

Les méthodes de test avec vecteurs prédéterminés

La méthode de génération prend en compte la structure ou la fonction du circuit.

Les approches *structurelles* s'appuient sur une analyse du schéma logique (schéma de portes ou de transistors) ou du schéma topologique (schéma des masques). Une connaissance approfondie du circuit est nécessaire.

Un ensemble de pannes possibles (hypothèses de panne) est défini sur ce schéma. Un ensemble de vecteurs de test détectant ces pannes est généré par des algorithmes dits "algorithmes de génération déterministe de vecteurs de test". L'exemple le plus connu est la méthode de génération des vecteurs de test de collages d'un circuit, fondée sur la notion de chemin sensible ou D-algorithme [Rot 68]. De telles méthodes sont inapplicables aux microprocesseurs pour deux raisons :

- La méconnaissance par l'utilisateur d'un schéma logique équivalent, l'établissement même d'un schéma logique équivalent pose des problèmes pour certaines technologies et de plus le schema est souvent modifié sans que l'utilisateur ne soit informé.

- La complexité des algorithmes [Par 82]. Si N est le nombre de portes du circuit, le temps de calcul pour faire la génération des vecteurs de test et la simulation des pannes est proportionnel au cube du nombre de portes $T = KN^3$. Il a été observé que le temps de calcul pour la simulation seule est proportionnel à N^2
  Pour un microprocesseur tel que le MC 68000 ou $N = 2 \times 10^4$, le temps de synthèse des vecteurs de test est de l'ordre de 10^{13} (si on estime à 1 μ sec le temps de base on obtient un temps de calcul de l'ordre de 100 jours !), le temps d'analyse de l'efficacité (simulation) est de l'ordre de l'heure ($T = K \times 10^{-1}$ heures)

Les approches *fonctionnelles ou comportementales* sont indépendantes de la réalisation physique du circuit et sont fondées sur la description de la fonction ou comportement du circuit.

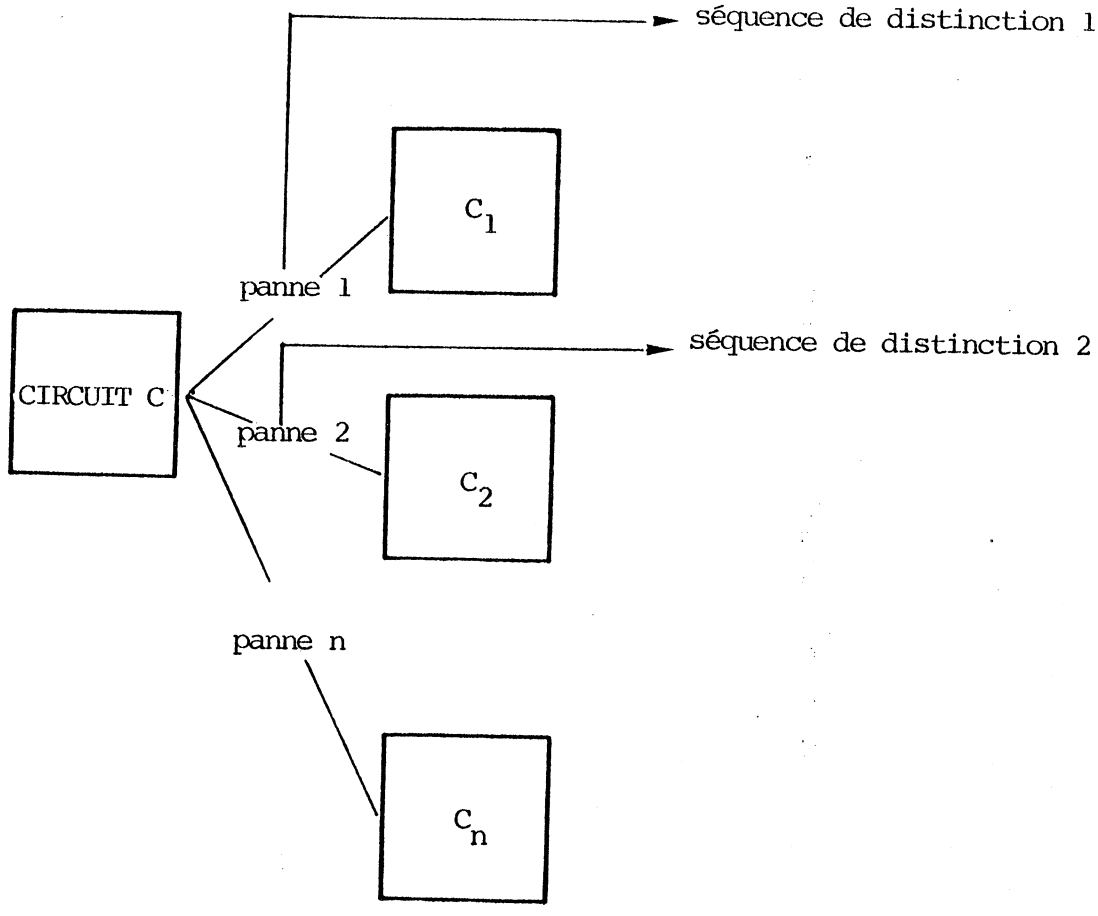


Figure 1

De telles approches proposées par Poage [Poa 63] ont été étendues aux circuits intégrés dans [Bel 81]. Leur complexité les rend totalement inapplicables aux microprocesseurs (il y aurait environ 60 000 machines fausses pour le MC 68000).

Une extension courante consiste à ne pas raisonner en termes de panne mais d'erreur. Il s'agit de partir d'un circuit juste, d'associer à l'ensemble des pannes considérées l'ensemble des erreurs produites puis de chercher des séquences distinguant les circuits justes des circuits avec erreur.

Les vecteurs de test sont alors :

- Soit exhaustifs, mais, dans le cas des circuits LSI le nombre d'états et d'entrées possibles rend impossible cette approche. En effet, si m est le nombre d'entrées et n le nombre de points de mémorisation du circuit, le nombre de vecteurs de test est alors 2^{m+n}. Un circuit LSI typique a 2^{180}, soit a peu près 10^{54}, états possibles [Nic 80]. En supposant que l'on a les vecteurs de test, et qu'ils sont appliqués à une fréquence de 1 MHZ, le temps de test est de 3×10^{39} années (l'âge de l'univers est estimé à $1,5 \times 10^{10}$ années).
- Soit fondés sur un principe de couverture de tous les modes de fonctionnement (sans chercher à générer toutes les données) ; il s'agit alors d'un test de conformité ou d'identification de fonctionnement.
- Soit fondés sur un principe de couvertures de pannes ou d'erreurs fonctionnelles.

Ce dernier cas pose un grand nombre de problèmes. En effet, si l'on veut retrouver en toute rigueur les raisonnements analytiques des approches structurelles, il faut considérer les pannes. Les différentes étapes sont alors :

- Description du fonctionnement juste du circuit.
- Liste de pannes considérées.
- Description des fonctionnements du circuit en présence de panne.
- Recherche d'une séquence de test ou séquence de distinction permettant de distinguer le circuit juste du circuit faux.

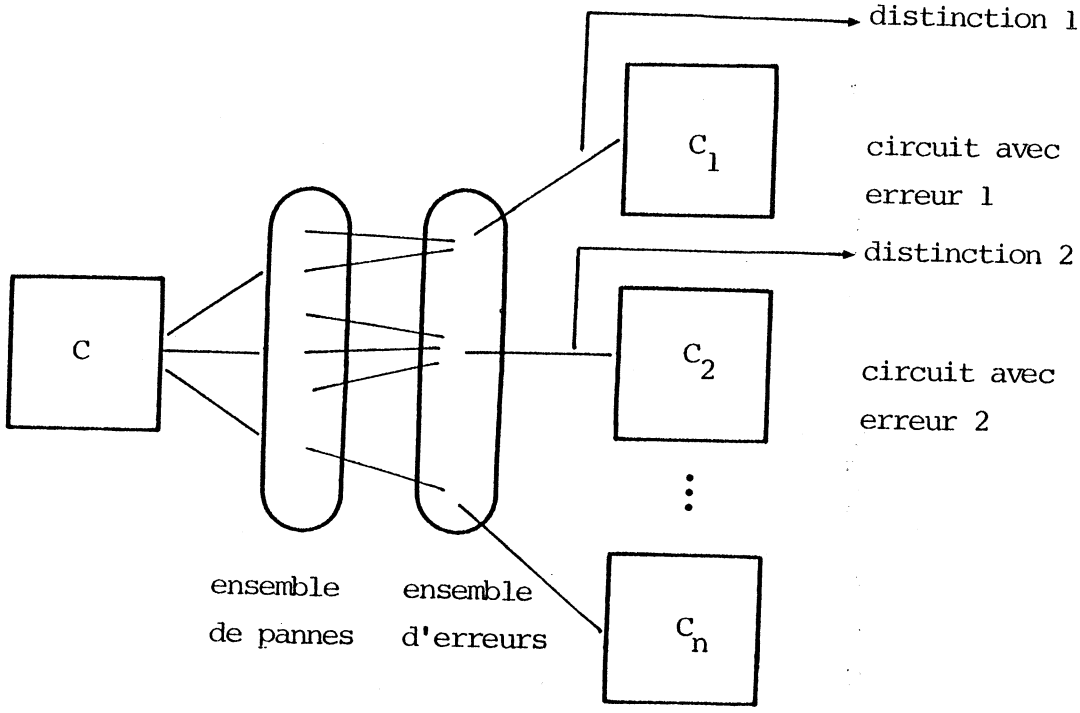


Figure 2

Un exemple de cette approche est celle proposée par Sridhar et Hayes [Hay 81] pour définir une méthode de test pour les microprocesseurs en tranches. Les auteurs exploitent une connaissance relativement détaillée de l'organisation interne de ce type de circuit. Le modèle proposé pour les microprocesseurs en tranches de 1 bit, consiste en l'interconnection d'un nombre réduit de blocs fonctionnels tels que accumulateurs, registres tampon, multiplexeurs et opérateurs (UAL, registre à décalage ...)

Les hypothèses d'erreur considérées sont :

- un mauvais fonctionnement change la table de vérité d'un circuit combinatoire mais il ne devient pas séquentiel,

- un mauvais fonctionnement change la table d'états d'un circuit séquentiel, mais le nombre d'états n'est pas augmenté .
- le système d'horloge n'est pas considéré comme susceptible de mauvais fonctionnement.

Les blocs composant le modèle sont de petite taille ; cela permet l'application de vecteurs de test exhaustifs pour détecter les erreurs définies. Les modules individuels sont donc testés exhaustivement, alors que leur interconnection est testée de manière non exhaustive .

Bien que le modèle et les procédures de génération des vecteurs de test soient formellement étendues à des processeurs en tranches de plus d'1 bit (2,4,...) et à des réseaux cellulaires de processeurs 1 bit, cette approche n'est malheureusement pas applicable aux microprocesseurs (8 et 16 bits) qui font l'objet de notre étude.

De toute façon, deux problèmes évidents se posent dans les approches basées sur la couverture d'erreurs :

- Le passage des pannes aux erreurs oblige à nouveau à considérer l'analyse des fonctionnements erronnés.
- L'ensemble des erreurs risque d'être d'une cardinalité comparable à celle des pannes.

Une solution tentante consiste à se donner à priori un ensemble d'erreurs de cardinalité "maitrisable" et qui de surcroit conduise à des séquences de distinction faciles à établir. Il s'agit alors de prendre des "hypothèses "d'erreurs fonctionnelles qui permettent de résoudre le problème, mais qui n'ont pas forcément un rapport avec la réalité.

Une méthode représentative de cette approche est celle proposée par Abraham et Thatte en [Tha 78][Tha 79].

Les microprocesseurs sont modélisés par un graphe orienté représentant les flots de données entre les registres internes pendant l'exécution des instructions. Les fonctions du microprocesseur sont divisées en cinq classes :

- fonction décodage des instructions,
- fonction sélection des registres,
- fonction de transfert de données,
- fonction de stockage de données,
- fonction de manipulation de données.

Des hypothèses d'erreurs fonctionnelles sont définies pour chacune des fonctions ; une analyse du comportement du microprocesseur en présence de ces erreurs permet de définir des algorithmes de test, chaque algorithme détectant une classe d'erreur particulière. Le problème est alors la détermination des séquences de test pour certaines classes d'erreurs. Notamment, les hypothèses d'erreur sur la fonction de décodage des instructions rendent le problème difficilement maitrisable dans le cas des microprocesseurs ayant un grand nombre d'instructions (microprocesseurs 16 bits par exemple). En effet ces hypothèses sont : lors de l'exécution de l'instruction Ij une autre instruction est exécutée à sa place, ou d'autres instructions sont activées en plus de Ij, ou encore aucune instruction n'est exécutée.

De telles approches et de telles hypothèses ont été à l'origine inspirées par les travaux de Robach et Saucier [Rob 78], mais dans ce travail un état du controle était materialisé par un élément de mémorisation discret (bascule) dans un séquenceur en logique discrete codé un parmi n . L'extension en technologie très integrée nous parait pas justifiable.

En [Lin 80] cette méthode est appliquée au microprocesseur INTEL 8086 mais les erreurs sur la fonction de décodage des instructions ne sont pas considérées.

A l'opposé de l'approche d'Abraham et Thatte se trouve celle proposée par Chiang et McCaskill [Chi 76]: le microprocesseur est divisé en modules accessibles à travers l'exécution des instructions adéquates. L'objectif du test n'est pas de couvrir un ensemble d'erreurs fonctionnelles, mais de détecter de mauvais fonctionnements par l'exécution de différentes séquences d'instructions chacune destinée à activer un des modules considérés. Ce concept est illustré à l'aide du microprocesseur 8080 d'INTEL. L'absence d'hypothèse d'erreurs est l'une des caractéristiques intéressantes de cette méthode. Une démarche similaire sera faite dans la suite de cette thèse, mais l'effort sera porté sur la formalisation de la description des microprocesseurs et sur l'étude du problème d'accès aux modules, cela en vue d'obtenir une méthode systématique et ainsi applicable à tout microprocesseur.

B - MISE EN OEUVRE DU TEST

Si on considère maintenant la mise en oeuvre du test on peut distinguer deux types de méthodes, l'une nécessitant des équipements de test spéciaux et l'autre non.

B -1 Test en environnement réel, sans équipement de test [Cla 79]

Le microprocesseur est testé dans son environnement "naturel" (microprocesseur avec mémoires ROM et RAM, contrôleurs des bus et périphériques) Le microprocesseur exécute un programme de test (chargé en RAM ou déjà existant en ROM). Le programme de test est conçu de manière à utiliser le plus grand nombre possible d'instructions dans des conditions de pire cas .

Cette méthode ne nécessite pas de matériel de test. Elle présente les inconvénients suivants :

- restriction de la commandabilité : seuls les signaux d'entrée utilisés par l'application seront commandés ; certaines configurations des entrées ne pourront pas être appliquées au microprocesseur.
- restriction de l'observabilité qui pourra être indirecte à travers d'autres unités, donc augmentation de la probabilité de masquage des erreurs
- restriction de l'espace d'adressage : la taille du programme de test est limitée par celle de la mémoire de l'application.

B -2 Test du microprocesseur nu, avec équipement de test [Hus 75][Cas 77]

a) Banc de test avec circuit de référence

Les vecteurs d'entrée (prédéterminés ou aléatoires) sont appliqués simultanément au microprocesseur sous test et à un microprocesseur qui sert de référence. Les sorties du circuit sous test sont comparées à celles du circuit de référence ; si une différence est détectée le diagnostic est alors circuit "mauvais", sinon le circuit est déclaré "bon".

Le matériel spécifique à cette technique est simple à développer (Fig 3).

Les principaux inconvénients sont la dépendance de la qualité du microprocesseur de référence, et l'impossibilité de déterminer la cause de l'erreur. De plus les erreurs de conception ou défauts de masque ne seront pas détectés car les deux circuits (circuit de référence et circuit sous test) réagiront de la même manière lors du test.

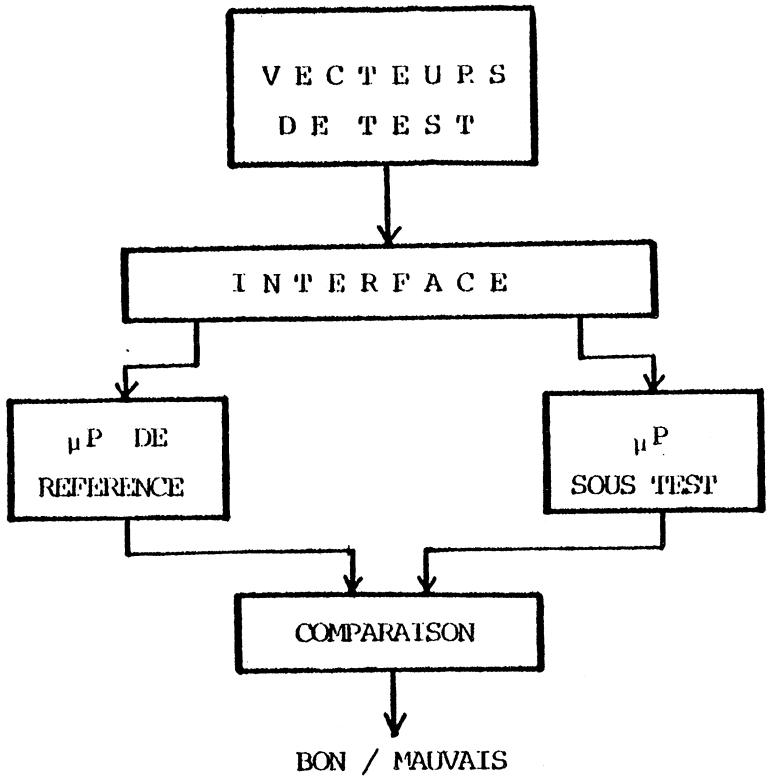


Figure 3 : Banc de test avec circuit de référence

b) Banc de test avec vecteurs de test et résultats préstockés

Les vecteurs de test ou les résultats attendus sont stockés dans la mémoire d'un système de test. Les résultats attendus sont obtenus par l'un des moyens suivants :

- application des vecteurs d'entrée à un circuit de référence,

- simulation (matérielle ou logicielle).

L'utilisation d'un circuit de référence présente les mêmes inconvénients que ceux de la méthode précédente, mais l'incertitude sur la qualité du circuit de référence peut être levée par comparaison des résultats obtenus à l'aide de plusieurs circuits réputés bons.

Le microprocesseur sous test est placé dans le système de test (testeur) qui, à l'aide des interfaces adéquates, applique les vecteurs d'entrée et compare, en fin de test, les résultats obtenus aux résultats préstockés (Fig 4).

Cette méthode est largement utilisée, elle permet de réaliser des tests logiques "complets". De plus l'utilisation de testeurs industriels performants (dits testeurs universels) permet aussi la mise en oeuvre de tests paramètriques statiques et dynamiques .

Quelques inconvénients sont :

- Les vecteurs de test étant préenregistrés leur modification au cours du test n'est pas facile à réaliser.
- Le prix des systèmes de test est élevé (de 1 MF à 4 MF).

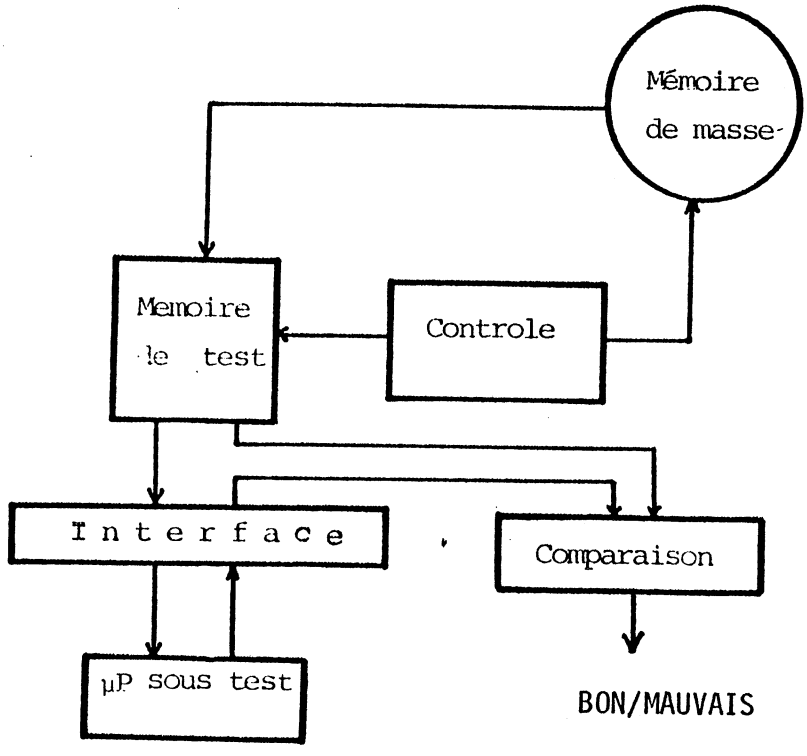


Figure 4 : Banc de test avec vecteurs et résultats préstockés

c) Banc de test avec génération en temps réel

Les vecteurs de test sont générés dans le langage machine du microprocesseur. Les vecteurs correspondant à une instruction sont envoyés au microprocesseur sous test lorsqu'il le demande (donc en temps réel). Simultanément le fonctionnement du microprocesseur sous test est émulé pour générer les sorties attendues avant que le microprocesseur sous test ne réponde (Les réponses peuvent aussi être précalculées, dans ce cas la méthode s'approche de la méthode précédente). Les réponses sont ensuite comparées et le test se continue par l'application d'une autre instruction (Fig 5).

Cette méthode semble très puissante compte tenu de sa flexibilité et de sa compacité. En effet, la génération algorithmique des vecteurs permet d'éviter leur stockage. Cependant le temps d'émulation peut rendre difficile son application lors du test de circuits rapides (nouvelles technologies).

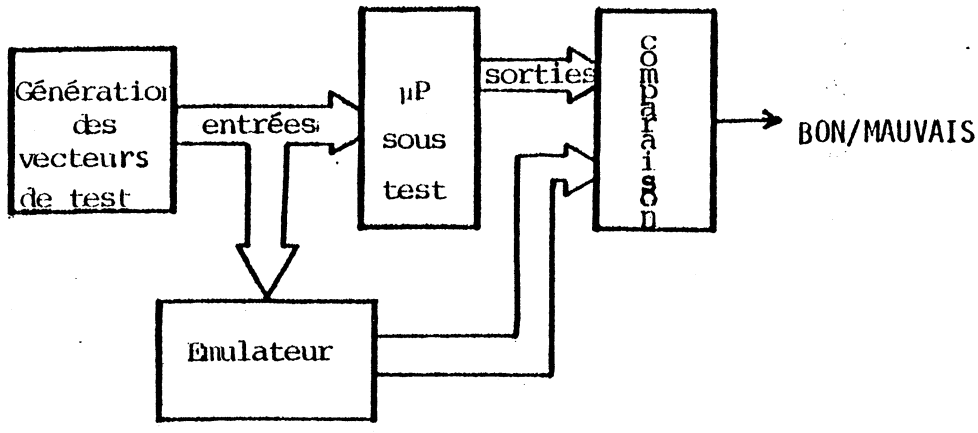


Figure 5 : Banc de test avec génération en temps réel

II - CHOIX D'UNE MÉTHODE ET D'UN ENVIRONNEMENT DE TEST

A - *METHODE PROPOSEE*

On propose ici une méthode de test de microprocesseurs basée sur une description comportementale. Cette méthode doit :

- être adaptable à tout microprocesseur,
- permettre une génération automatique de programmes de test à partir de cette description.

La description comportementale s'appuie sur les informations contenues dans le manuel utilisateur :

- le jeu d'instructions,
- l'ensemble des éléments de mémorisation et des opérateurs,
- les signaux de dialogue.

Trois étapes de test seront appliquées au microprocesseur :

- un *test de conformité* qui permettra de vérifier le bon déroulement des instructions,
- un *test de balayage* qui permettra de vérifier le bon fonctionnement des différents blocs fonctionnels,
- un *test des signaux complémentaires* (signaux de dialogue qui ne sont pas activés par les instructions)

La première partie du test dit de conformité est une vérification de la circulation du flot de données au cours de l'exécution d'une instruction *indépendamment* de toute hypothèse de pannes ou d'erreurs fonctionnelles.

Ce type de test se rapproche d'un test d'identification, d'algorithme ou de logiciel. Nous nous sommes à dessein refusé à faire reposer l'ensemble du programme de test sur des suppositions trop arbitraires de mauvais fonctionnement (erreur fonctionnelle). On ne cherche pas, en particulier, à distinguer un microprocesseur "juste" des microprocesseurs "faux".

Par contre, dans un deuxième module, dit test de balayage, on propose de partitionner le microprocesseur en blocs et d'activer ces blocs par des opérandes de test spécifiques. L'adjonction de tels modules permet de prendre en compte de façon modulaire les connaissances que l'on peut avoir, ou que l'on suppose avoir, de la structure des blocs, ainsi que d'opérandes de test déterministes.

Le dernier type de test complète les deux premiers types. Il s'agit du test des signaux d'interface non testés précédemment et nécessitant un environnement et un équipement spéciaux.

Les différents problèmes abordés sont illustrés à l'aide des microprocesseurs courants tels que MC 6800, MC 68000, Z 80, ... Les caractéristiques de ces microprocesseurs telles qu'elles sont présentées dans les manuels d'utilisation, sont données dans l'annexe 5 .

B - CHOIX D'UN ENVIRONNEMENT DE TEST

B -1 Types d'environnements de test

L'environnement de test a des conséquences importantes sur la structure des programmes de test et encore plus sur leur génération automatique.

On distingue trois solutions :

- environnement "classique",
- environnement "minimal",
- environnement "hybride".

a) Environnement classique

L'équipement de test (testeur) émet les séquences d'entrées vers le microprocesseur testé, et observe l'état de ses sorties (Fig 6). Ces testeurs permettent la commande indépendante des différentes broches d'entrée du circuit sous test (y compris les entrées d'horloge), la fréquence d'émission des vecteurs d'entrée est alors imposée par le système de test. Ces testeurs , dits aussi "testeurs universels" sont d'une grande flexibilité d'utilisation et fournissent les moyens logiciels et matériels pour réaliser le test de tout circuit.

Dans le cas du test comportemental de microprocesseurs l'utilisation d'un tel environnement se heurte aux difficultés suivantes :

- Chaque instruction du programme de test doit être transformée en séquence de valeurs des broches d'entrée (analyse des instructions au niveau du cycle).
- Le réordonnancement des mots du programme de test pour l'obtention des séquences d'entrée : en effet, un microprocesseur qui a la possibilité d'anticipation ou de "pipe lining" entre instructions, n'effectue pas les lectures mémoire dans l'ordre logique fonctionnellement apparent.

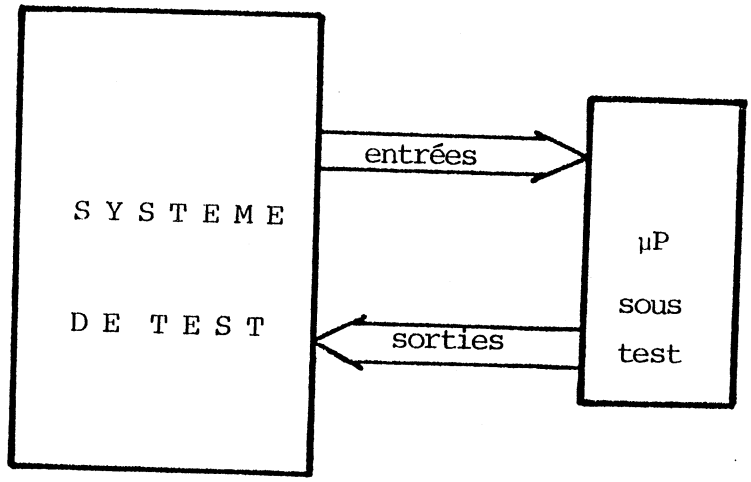


Figure 6 : Environnement classique

b) Environnement "minimal"

Le système de test est composé d'une mémoire RAM et de l'interface adéquate permettant au microprocesseur testé d'exécuter un programme de test en "fonctionnement normal" : une fois la mémoire de test chargée, le microprocesseur est lancé et exécute le programme de test, lisant le programme de test et ses opérandes dans la mémoire de test et stockant les résultats dans cette même mémoire (Fig 7).

Ce type d'environnement permet l'exécution de programmes de test sans transformations ou réordonnancement préalables.

La commande et l'observation respectivement des signaux d'entrée et de sortie se fait via la mémoire lors de l'exécution du programme de test, ceci entraîne les inconvénients suivants :

- la mémoire est partagée entre le programme, les données et les résultats, ce qui impose une gestion complexe de l'espace mémoire,
- certains des signaux d'entrée ou de sortie du microprocesseur sous test ne sont pas concernés par le fonctionnement dit "normal". Ces signaux ne peuvent pas être pris en compte pour le test sans ajouter du matériel spécifique et sans résoudre des problèmes difficiles de synchronisation. Le nombre de ces signaux spécialisés devient de plus en plus important (microprocesseurs 16 bits) ; leur test est donc nécessaire.

Cette solution (environnement minimal) n'est pas adaptée au problème du test des microprocesseurs (8 et 16 bits) avec génération automatique des programmes de test. Cependant elle a été adoptée dans un premier temps lors de la réalisation expérimentale (génération semi-automatique de programmes de test pour microprocesseurs 8 bits) proposée dans la Section II

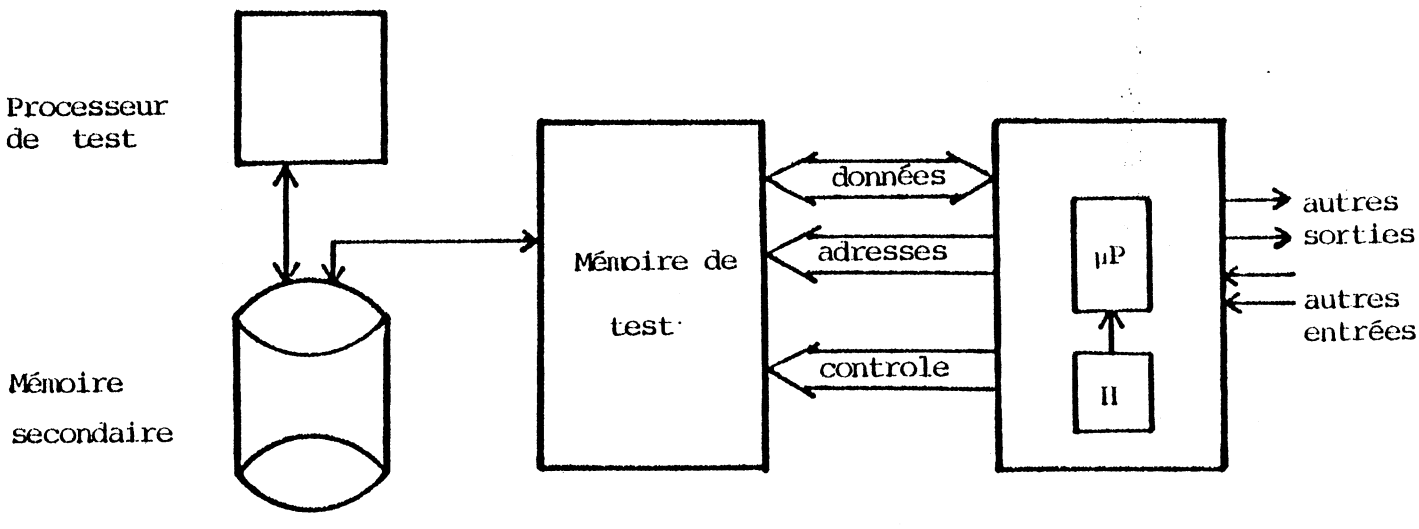


Figure 7 : Environnement minimal

c) Environnement "hybride"

La solution adoptée, l'environnement dit "hybride", est un système de test dans lequel :

- Le microprocesseur testé exécute un programme de test stocké dans une mémoire RAM (mémoire de test) mais il n'accède à cette mémoire qu'en lecture.
- Chaque broche de sortie du microprocesseur est observée et sa valeur est mémorisée aux instants où elle est valide.
- Un matériel spécifique est prévu pour le test des signaux.

La figure 8 présente les principales caractéristiques de l'environnement choisi.

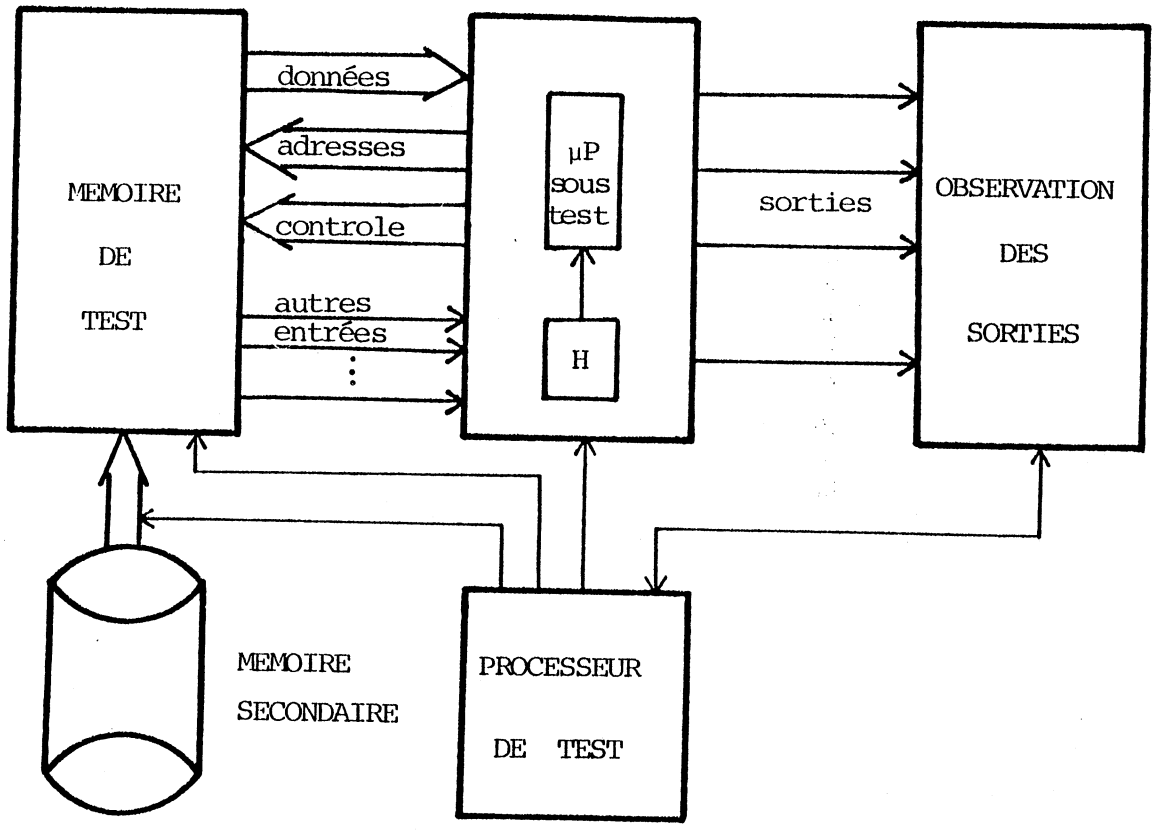


Figure 8: Environnement hybride

- *La mémoire secondaire* contient les programmes de test ainsi que les opérandes de test ; le *processeur de test* les transfère dans la mémoire de test.

- *La mémoire de test* est accédée par le microprocesseur sous-test *seulement en lecture* (les signaux d'écriture provenant du microprocesseur sous test ne seront pas pris en compte). Elle contient les programmes de test et ses opérandes ainsi que les données permettant la commande des différents signaux de contrôle du microprocesseur (de type entrée).

- *La carte de test* est l'interface entre le microprocesseur sous test et l'environnement. Elle fournit un système d'horloge adéquat.

- *L'observation des sorties* permet le stockage des valeurs des broches de sortie du microprocesseur, aux instants où elles sont valides. L'observation des sorties peut être réalisée par une mémoire ou dans des registres d'analyse de signature [Gor 77][Mit 77].

L'environnement de test (bus, mémoire, observation des sorties ...) sera conçu de manière à ce que tout microprocesseur *8 et 16 bits* puisse être testé. Seule la carte interface sera spécifique à chaque microprocesseur.

B -2 *Conséquences du choix d'un environnement hybride*

L'utilisation d'un tel environnement "hybride" fait que :

- toute broche d'entrée du microprocesseur sous test est commandable,
- toute broche de sortie est observable.

L'approche choisie permet :

- de simplifier la gestion mémoire,
- d'observer toutes les valeurs des sorties du microprocesseur, et donc facilite la génération automatique des programmes de test.

De plus, l'environnement choisi fournit la possibilité de compacter les résultats de test par utilisation de registres de signature.

La génération de programmes de test tiendra compte des caractéristiques de cet environnement. Cependant, les programmes de test générés pourront être utilisés dans un environnement de test classique, moyennant leur transcription en séquence d'entrée. Cette transcription dépend d'une part du microprocesseur testé, d'autre part du type de testeur utilisé.

III - DESCRIPTION DES MICROPROCESSEURS

La généralisation de la méthode de test à tout microprocesseur passe par la définition d'une description normalisée des microprocesseurs. Cette description doit être un outil de représentation :

- par lequel tout microprocesseur puisse être décrit,
- sur lequel la méthode de test comportemental puisse être appliquée et les programmes de test correspondants puissent être générés automatiquement.

A - DESCRIPTION PROPOSEE

La description est déduite du manuel utilisateur du microprocesseur et comporte deux parties :

- la description du matériel,
- la description du fonctionnement.

A -1 Dans la partie *description du matériel* sont données les informations relatives

- à l'architecture du microprocesseur (format des données, éléments de mémorisation, unités fonctionnelles, ...),
- aux différents signaux d'entrée et de sortie du microprocesseur (signaux d'interruption, bus adresse et données, signaux de contrôle..).

A -2 Dans la partie *description du fonctionnement* sont portées les informations décrivant les fonctions du microprocesseur, ce sont :

- le jeu d'instructions
- le comportement face aux signaux d'entrée asynchrones.

Lors de l'exécution d'une instruction, des éléments de mémorisation et des blocs fonctionnels sont activés par l'unité de contrôle pour réaliser les traitements correspondants à l'instruction considérée. Le détail de ces activations n'est pas toujours disponible (structure interne du microprocesseur inconnue) à l'utilisateur.

* *Chaque instruction est décrite par l'ensemble d'activations apparentes* telles que :

- transfert entre éléments de mémorisation,
- transfert entre éléments de mémorisation et l'extérieur (mémoire ou périphérique),
- manipulation de données,
- activations (chargement, rangement ou manipulation) conditionnées par la valeur d'éléments de mémorisation,

- itération d'après la valeur d'un ou plusieurs éléments de mémorisation,
- mise à un état déterminé d'un ou plusieurs signaux de sortie du microprocesseur.

Les activations correspondant à la recherche de l'instruction (code opération, adresse explicite ou valeur immédiate) étant communes à toute instruction , elles ne sont pas décrites dans le modèle .

**Le comportement face aux signaux d'entrée asynchrones* sera décrit par le même moyen. Cependant la grande diversité des signaux rend difficile la normalisation de la description. On ne décrit par ce moyen que les signaux d'entrée ayant des conséquences fonctionnelles sur le matériel défini dans la partie description du matériel, les autres seront étudiés comme cas particuliers .

B - DEFINITIONS

Pour une instruction on définit deux ensembles : l'ensemble des sources et l'ensemble des puits .

Définition 1

S est une *source* pour l'instruction I si S est support d'une information utilisée (ou traitée) pendant l'exécution de I.

On distingue :

- les *sources externes* : le support de l'information se trouve à l'extérieur du µP (mémoire ou périphérique).
- les *sources internes* : le support de l'information traitée, est un (ou un ensemble d'éléments de mémorisation du microprocesseur.

Remarque : On ne considère l'instruction qu'après la phase de "fetch" (recherche de l'instruction). Les informations contenues dans l'instruction (opérandes immédiats, adresse explicite) seront appelées *sources immédiates*.

Exemples : instructions tirées du jeu d'instruction du MC 6800 [Annexe 5]

a) instruction LDAA # n chargement de l'accumulateur A avec la valeur immédiate n : n est une source immédiate.

b) instruction LDAA n,X chargement de l'accumulateur A avec le mot mémoire dont l'adresse est obtenue par addition du contenu du registre X et de la valeur immédiate n :

- l'emplacement mémoire d'adresse (X) + n est une source externe.
- le registre X est une source interne, la valeur n est une source immédiate.

Définition 2

P est dit *puits* pour l'instruction J si P est un support d'information dont la valeur est ou peut être modifiée par l'exécution de J.

On distingue de même les puits internes et externes.

Remarque : Le compteur programme (noté PC)est modifié par l'exécution de toute instruction. Il ne sera considéré explicitement comme puits que dans le cas de rupture de séquence.

Exemple :

a) instruction ABA d'addition des accumulateurs A et B, stockage du résultat dans A.

- A est un puit interne car il mémorise le résultat de l'opération d'addition.

- CC registre des indicateurs est un puit interne car certains des bits qu'il comporte sont modifiés par l'exécution de l'instruction (indicateurs N, Z, V, C)

b) instruction STAA m rangement du contenu de l'accumulateur A à l'adresse m.

- m est une source immédiate.
- A est une source interne
- l'emplacement mémoire d'adresse m est un puits externe
- CC est un *puits interne*(modification de bits N, Z).

c) instruction JSR m de branchement au sous programme d'adressse m, avec sauvegarde du compteur ordinal PC dans une pile en mémoire pointée par SP.

- source interne PC et SP.
- puits interne PC , SP ,
- les emplacements mémoire d'adresse SP, SP -1 sont des puits externes (sauvegarde du PC).

Définition 3

Une source est dite de *type adresse* si son contenu sert à calculer l'adresse d'un support mémoire effectivement accédé au cours de l'instruction, dans le cas contraire elle est appelée source de *type donnée* .

Exemple :

a) Dans l'instruction LDAA n,X X est une source interne de type adresse.

b) Dans l'instruction JMP n,X de branchement à l'adresse donnée par (X) + n, X est une source de type donnée car le mot mémoire d'adresse (X) + n n'est pas effectivement accédé.

Les éléments de mémorisation E M du microprocesseur peuvent alors être classés, pour l'instruction I, en trois ensembles :

- S_{adr} = {E M qui sont sources de type adresse}
- S_{data} = {E M qui sont sources de type donnée}
- P = {E M qui sont puits}

Les sources immédiates seront aussi classées d'après ce critère, en sources immédiates de type adresse ou de type donnée.

C - LANGAGE DE DESCRIPTION

La description adoptée peut être présentée sous forme d'un programme écrit dans un langage spécifique. Ce langage doit donc permettre :

- La définition de variables telles que :
  . éléments de mémorisation : registres, bascules,...
  . signaux d'entrée et de sortie.
- La définition d'opérateurs tels que :
  . UAL (addition, soustraction, et, ou, complément,...)
  . MULT, DIV, ...
  . .

- La description des activations apparentes correspondantes aux instructions et signaux asynchrones.

Des langages structurés haut niveau (type RTL par exemple) sont adaptés à ces besoins.

On se limite ici à présenter un exemple de langage de description et son utilisation pour décrire des instructions représentives du MC 6800.

La syntaxe du langage de description adopté est décrite en [VEL 82].

Dans le paragraphe 6 les caractéristiques importantes pour la génération automatique sont évoquées.

C -1 : Description partielle d'un microprocesseur

Pour la clarté de l'exposé la grammaire du langage de description n'est pas explicitée. Cet exemple étant destiné plutôt à illustrer les notions définies dans les paragraphes précédents on s'est volontairement limité à la description partielle d'un microprocesseur qui est loin d'avoir la complexité des µP futurs : il s'agit du microprocesseur 8 bits MC 6800. Cependant des enseignements pourront être tirés de cet exemple en ce qui concerne le type de langage et de description qui s'adapteront le mieux au traitement de n'importe quel microprocesseur.

a) Caractéristiques du langage

Il permet de définir :

- des variables telles que :
  - reg_p élément de mémorisation de p bits
  - mem(adr) mot mémoire d'adresse adr
  - bit i de reg_p un bit à l'intérieur d'un élément de mémorisation

- des opérateurs arithmétiques et logiques :

  . ADD, SUB, ..., ET, OU, ...,

- des signaux d'entrée asynchrones : S_e

- des signaux de sortie : S_s

Les activations correspondant à l'exécution d'une instruction sont des séquences écrites à l'aide des primitives :

- de transfert : $EM_1 \leftarrow EM_2$

- de manipulation EM_1 op (op opération unaire)

  EM_1 op'EM_2 (op' opération binaire)

- conditionnelles : Si COND alors activation.

Les valeurs immédiates et adresses explicites ont été notées

- V_p valeur immédiate sur p bits

- adr_q adresse explicite sur q bits

Dans le cas où un des opérandes provient de la mémoire on note Mem(adr) l'adresse pouvant être explicite ou une expression représentant le mode d'adressage (à l'aide des opérateurs "+" ou "-" appliqués aux registres et valeurs concernés).

b) Application au microprocesseur MC 6800

La table 1 montre un ensemble significatif de fonctionnements du MC 6800 et la description du matériel mis en jeu.

DECLARATIONS

reg_8 = (A, B, CC)

reg_{16} = (X, SP, PC)

CC = (1, 1, H, I, N, Z, V, C)

OPRT = (ADD, ADC, INC)

Sgnx = ($\overline{IRQ}$, BA, TSC, D_0, D_1 ... D_7, A_0, A_1 ... A_{15})

FONCTIONNEMENT

I 1	:	LDAA	n	$A \leftarrow v_8$
I 2	:	LDS	m	$SP \leftarrow Mem(adr_{16})$
I 3	:	STAB	n	$Mem(adr_8) \leftarrow B$
I 4	:	LDAA	n,X	$A \leftarrow Mem(X + v_8)$
I 5	:	TAP		$CC \leftarrow A$
I 6	:	PULA		$A \leftarrow Mem(SP + 1)$; $SP \leftarrow SP + 1$
I 7	:	PSHB		$Mem(SP) \leftarrow B$; $SP \leftarrow SP - 1$
I 8	:	CLRI		$I \leftarrow 0$
I 9	:	SEI		$I \leftarrow 1$
I 10	:	ADDA	m	$A \leftarrow A$ add $Mem(adr_{16})$
I 11	:	ADC	n,X	$A \leftarrow A$ adc $Mem(X + adr_8)$ adc C
I 12	:	INCA		$A \leftarrow A$ inc
I 13	:	BNE	d	Si Z = 0 alors $PC \leftarrow PC + V_8$
I 14	:	JSR	m	$Mem(SP) \leftarrow PC$; $SP \leftarrow SP - 2$; $PC \leftarrow v_{16}$
I 15	:	RTS		$PC \leftarrow Mem(SP + 1)$; $SP \leftarrow SP + 2$

Table 1 : Description partielle du MC 6800

I 16 : WAI

Mem(SP) ← PC
Mem(SP-2) ← X
Mem(SP-4) ← A
Mem(SP-5) ← B
Mem(SP-6) ← CC
SP ← SP - 7
BA ← 1 ↑

S 1 : $\overline{IRQ}$

Si (I = 1) et ($\overline{IRQ}$) alors
    Mem(SP) ← PC
    Mem(SP-2) ← A
    Mem(SP-3) ← B
    Mem(SP-4) ← CC
    SP ← SP - 7
    PC ← Mem(FFF8)
fsi

S 2 : TSC

Si TSC alors
    $D_0, D_1, \ldots, D_7$ ← HI
    $A_0, A_1, \ldots, A_{15}$ ← III
fsi

Suite Table 1 : Description partielle du MC 6800

D - REMARQUES

1) Les instructions de transfert $I_1, \ldots, I_7$ sont décrites à l'aide de la primitive "←". On peut associer une (cas des instructions $I_1, \ldots, I_5$) ou plusieurs (instructions I_6, I_7) primitives agissant sur des éléments de mémorisation ou des bits à l'intérieur d'un élément de mémorisation (instructions I_8 et I_9)

2) Les instructions de manipulation de données I_{10}, I_{11}, I_{12} ont été décrites avec la notation infixée. L'opérateur, tel qu'il est défini dans les DECLARATIONS, est encadré du ou des opérandes.

3) L'instruction de branchement conditionnel I_{13} est décrite à l'aide de la primitive si cond alors. En ce qui concerne les instructions de branchement et retour de sous-programme (I_{14} et I_{15}) les primitives de transfert sont utilisées pour décrire la sauvegarde du compteur programme et/ou chargement de la nouvelle valeur du compteur programme. Les modifications intermédiaires éventuelles que peut subir le registre SP, n'ont pas été prises en compte. Seule la modification globale (représentant sa valeur après exécution de l'instruction en fonction de la valeur avant exécution de l'instruction) a été décrite.

4) La sauvegarde du compteur programme (instruction I_{14}, I_{16}) est décrite à l'aide d'une seule primitive de transfert, bien que dans la réalité ceci nécessite deux accès mémoire (PC poids forts, PC poids faibles).

5) L'instruction I_{14} est décrite par les transferts correspondant aux sauvegardes réalisées, plus l'activation du signal de sortie BA (noté BA←1↑).

6) Des signaux peuvent aussi apparaître comme opérandes dans des primitives. Tel est le cas pour S_1 et S_2, descriptions du comportement face aux signaux asynchrones $\overline{IRQ}$ et TSC. Dans le cas de S_2 le signal TSC (contrôle trois états) permet la mise en haute impédance (HI) des bus données et adresses.

7) La modification des bits du registre CC doit être indiquée pour chaque instruction (équations booléennes pour chaque indicateur). On a omis CC de la description pour la clarté de l'exposé .

Le type de description proposée dans l'exemple donné dans la Table 1 permet d'établir aisèment les ensembles de sources et de puits définis en II.C.

En effet, pour une instruction donnée les parties gauches des affectations sont les *puits*. Les puits externes sont explicitement indiqués par Mem (adresse).

En ce qui concerne les sources, deux cas sont possibles :

- Les *sources de type adresse* sont les variables qui font partie de l'expression d'une adresse en mémoire.
- Les *sources de type donnée* apparaissent dans les parties droites des affectations ou interviennent dans la condition des primitives si alors.

Exemples :

1) I 4 : LDAA n,X — A ← Mem(X + V_8)

Sources	V_8	source immédiate
	X	source interne de type adresse
	Mem(X + V_8)	source externe
Puits	A	puits interne

2) I 7 : PSHB — Mem(SP) ← B ; SP ← SP-1

Sources	SP	source interne de type adresse
	B	source interne de type donnée
Puits	SP	puits interne
	Mem(SP)	puits externe

3) I 16 : WAI

Sources	SP	source interne de type adresse
	A,X,B,CC,PC	sources de type donnée
Puits	SP	puits interne
	Mem(SP),...,Mem(SP-6)	puits externes
	Le signal BA est un puits externe	

IV - ACCÈS AUX ÉLÉMENTS DE MÉMORISATION

A - *INTRODUCTION : DEROULEMENT DE TEST*

Dans l'environnement de test défini (§ II.B) les trois étapes de la méthode de test : *conformité, balayage et signaux complémentaires*, sont mises en oeuvre par trois programmes stockés dans la mémoire secondaire. Le processeur de test les charge successivement dans la mémoire de test (éventuellement découpés en modules), lance leur exécution et en fin d'exécution compare les résultats obtenus avec des résultats préétablis (obtenus après exécution des mêmes programmes par un microprocesseur réputé bon).

Chacun des trois programmes est composé de modules élémentaires comportant les phases suivantes :

- Initialisation
- Stimulation
- Observation

A -1 Pendant la *phase d'initialisation* des données de test sont chargées dans des éléments de mémorisation du microprocesseur sous test.

A -2 Pendant la *stimulation* une séquence d'entrées spécifiques est appliquée aux entrées du microprocesseur.

A -3 Pendant la *phase d'observation* le contenu des éléments de mémorisation est émis du microprocesseur sous test vers l'environnement de test.

Les phases d'initialisation et d'observation seront mises en oeuvre par l'exécution, par le microprocesseur sous test, des instructions de transfert de données adéquates.

Compte tenu de la diversité des microprocesseurs auxquels sera appliquée la méthode, on étudie d'abord le problème de l'accès aux éléments de mémorisation. Ce problème sera abordé en trois étapes :

- Etude de la commande et de l'observation des éléments de mémorisation.
- Analyse et classification des instructions de transfert.
- Stratégie de construction des séquences d'initialisation et d'observation.

B - COMMANDE ET OBSERVATION DES ELEMENTS DE MEMORISATION

Parmi les éléments de mémorisation internes d'un microprocesseur, notés EM, on distingue :

- les EM directement observables,
- les EM indirectement observables.

B -1 EM directement observables

Il s'agit des éléments de mémorisation dont le contenu peut être observé à l'extérieur par exécution d'une séquence d'instructions appelée *séquence d'observation directe.*

L'exécution de la séquence peut, soit provoquer le transfert du contenu de l'élément de mémorisation à observer, soit émettre vers le monde extérieur des valeurs caractéristiques de la valeur de l'élément de mémorisation à observer.

Exemple 1 :

On considère le MC 6800 de Motorola.

- l'accumulateur A est directement observable par la séquence de longueur 1 : STAA adr
- le registre de code condition est directement observable par la séquence de longueur 2 :
  . TPA
  . STAA adr
- le compteur programme est directement observable par l'exécution de toute instruction, car son contenu transite par les lignes adresses lors du "fetch" de l'instruction. Il n'y aura donc pas explicitement de séquences d'observation du compteur programme.

Exemple 2 :

Exemple hypothétique d'observation directe mais complexe :

Supposons que le contenu de certaines bascules de condition ne peut être transféré vers l'extérieur. Le contenu de ces bascules sera observé par une série de branchements.

Soit C une de ces bascules ; brc, un branchement conditionnel suivant la valeur de cette bascule. La séquence d'observation est :

```
ad    brc  ad"
ad'   suite
.
ad"
```

Suivant la valeur de C, on a des valeurs différentes qui transitent sur le bus adresse (observation implicite de PC).

C	bus adresse
0	ad ad'
1	ad ad"

B -2 EM indirectement observables

Il s'agit des éléments de mémorisation dont le contenu (ou ses conséquences) peut être observé au point d'observation, mais uniquement avec l'intervention des signaux de dialogue.

Exemple pour le Z80 :

bascule IFF1 de masquage des interruptions

IFF1 = 0 interruptions masquées

observation PC ← PC + 1

IFF1 = 1 interruptions non masquées

observation PC ← adr de traitement de l'interruption

Dans les deux cas, il est nécessaire d'activer le signal d'interruption masquable IRQ et l'observation est faite en observant la valeur de PC.

B -3 EM directement commandables

On définira de même les éléments de mémorisation directement commandables ; le contenu de ces éléments peut être amené du monde extérieur par l'exécution d'une séquence d'instructions dite *séquence de commande directe.*

Exemple : (Z80)

- Le registre B est commandé par la séquence de longueur 1 :

```
LD   B, data
```

- Le registre interruption I, l'accumulateur A' du banc de registres secondaires et le registre d'indicateurs F sont commandables par les séquences de commande directe suivantes :

```
I    LD   A, data
     LD   I, A

A'   EX
     LD   A, data
     EX

F    LD   SP, adr
     POP  AF
```

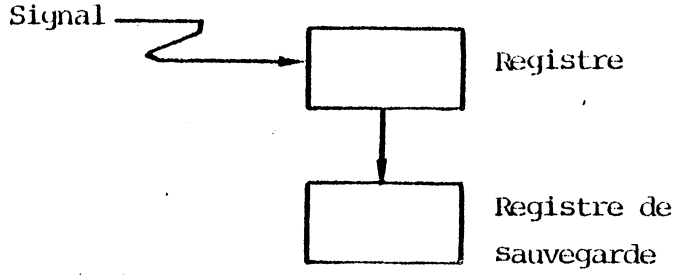
PC est commandé par JP adr

B -4 EM directement commandables

Ce sont les éléments de mémorisation pour lesquels la donnée provenant du monde extérieur ne peut être amenée qu'en faisant intervenir des signaux de dialogue.

Exemple :

Des registres servant à la sauvegarde d'un état lorsque certaines configurations de signaux de dialogue sont présentes.



C - ANALYSE DES INSTRUCTIONS DE TRANSFERT

On s'intéresse ici à l'étude et classification des instructions de transfert par rapport aux éléments de mémorisation qui sont directement commandables et observables. On étudie particulièrement les registres internes du microprocesseur.

C -1 Définitions préliminaires

Une instruction de transfert est définie de manière classique par :

- l'origine de l'information,
- la destination de l'information.

a) Considérons un registre R. Les instructions de transfert affectant ce registre peuvent être classées dans un premier temps en instructions de lecture ou d'écriture.

* Une instruction de transfert est une *instruction de lecture* pour R, si pendant son exécution la donnée stockée dans R est transférée dans un autre registre ou vers le monde extérieur.

* Une instruction de transfert est une *instruction d'écriture* si pendant son exécution une donnée provenant d'un autre registre ou du monde extérieur est stockée dans R.

Si on considère la description donnée en (III B) les instructions de *lecture pour R* sont donc les instructions de transfert dans lesquelles R apparaît comme *source* dans la description. Les instructions *d'écriture pour R* sont celles pour lesquelles R apparaît comme *puits*.

b) Une instruction de transfert affectant le registre R est une *instruction composée* si elle utilise un autre registre que R.

On peut distinguer deux cas :

- L'origine ou la destination est un autre registre (et non le monde extérieur),
- L'adresse de l'origine ou de la destination en mémoire externe est déterminée par le contenu d'un ou plusieurs registres.

Ces deux cas ne s'excluent pas mutuellement.

Une instruction qui n'est pas composée sera dite *instruction simple*.

Dans la description adoptée les instructions simples sont celles pour lesquelles :

- la seule source est l'extérieur et la destination de l'information est R : écriture simple de R (chargement avec une valeur immédiate ou avec adresse explicite)

ou

- la seule source est R et la destination de l'information est l'extérieur : lecture simple de R (rangement en mémoire à une adresse explicite)

Les autres instructions de transfert sont des *instructions composées*.

c) On dira qu'une instruction de transfert affectant le registre R a un *effet de bord* sur l'état du microprocesseur, si pendant son exécution, le contenu de registres autres que R,est modifié.

Les registres modifiés appartiennent par définition à l'ensemble des puits P de l'instruction considérée. Ce sont notamment :

- Des registres destination de l'information, autres que R.
- Des registres qui sont aussi sources pour l'instruction considérée, dont le contenu est modifié (incrémenté, decrémenté, mis à jour..) au cours de l'exécution de l'instruction. Par exemple, le registre SP (pointeur de pile) lors des instructions utilisant la pile.
- Des registres qui ne sont pas sources mais dont le contenu est modifié comme conséquence de l'exécution de l'instruction. Par exemple le registre des indicateurs dans le cas d'une instruction de transfert affectant certains des indicateurs d'après la valeur de la donnée transférée.

Parmi les puits d'une instruction, on distingue donc :

- Les puits externes éventuellement, notés EXT
- Les puits de type destination de l'information tranférée, notés P dest.
- Les autres puits qui sont des puits auxiliaires (par exemple pointeur de pile et registre d'état), notés P aux.

Les instructions de transfert, affectant un registre R, ayant un effet de bord, sont alors :

- Les instructions de lecture de R pour lesquelles $P \neq$ ext.
- Les instructions d'écriture de R pour lesquelles $P \supset R$ et $P \neq R$

d) Ensembles associés à une instruction de transfert

A toute instruction de transfert on associe quatre ensembles d'éléments de mémorisation notés respectivement S_{adr}, S_{data}, P_{dest} et P_{aux} :

- S_{adr} est l'ensemble de sources de type adresse
- S_{data} est l'ensemble de sources de type donnée
- P_{dest} est l'ensemble de puits de type destination
- P_{aux} est l'ensemble de puits auxiliaires
- l'extérieur noté EXT, peut faire partie de l'ensemble de sources, et/ou de puits.

C -2 Exemple

On considère le microprocesseur Z 80 dont les renseignements sont donnés en Annexe 5 .

Considérons l'accumulateur A :

- L'instruction LD A,n de chargement de la valeur n dans l'accumulateur A est une instruction simple d'écriture pour A. L'instruction LD (nn),A de rangement de A en mémoire à l'adresse nn est une instruction simple de lecture.
- L'instruction LD A,(HL) de chargement de A à partir de la mémoire pointée par la paire de registres H,L est une instruction d'écriture composée pour A. Les registres H et L étant les sources de type adresse.

- L'instruction POP AF de chargement de l'accumulateur A et du registre F avec les mots mémoire aux adresses (SP) + 1 et (SP), est une instruction d'écriture composée pour A (le pointeur de pile SP étant la source de type adresse) et présente un effet de bord pour SP et F. En effet pendant le déroulement de cette instruction F reçoit la donnée rangée en mémoire à l'adresse SP et le registre SP est mis à jour. On a donc :

$$S_{adr} = \{SP\}$$

$$S_{data} = \{\emptyset\}$$

$$P_{dest} = \{A,F\}$$

$$P_{aux} = \{SP\}$$

- Si l'on considère maintenant le registre I. L'instruction LD A,I de chargement de l'accumulateur A à partir du registre I, est une instruction de lecture pour I. Elle est composée car le monde extérieur n'est ni source ni destination. Pour cette instruction on a les ensembles :

$$S_{adr} = \{\emptyset\}$$

$$S_{data} = \{I\}$$

$$P_{dest} = \{A\}$$

$$P_{aux} = \{\emptyset\}$$

Les figures 9-a, ... 9-e illustrent ces 4 cas.

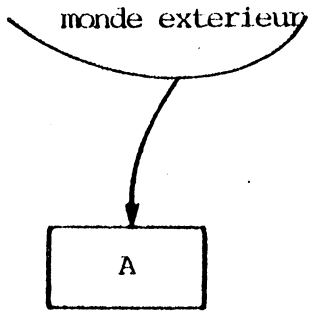


Figure 9-a: Ecriture simple

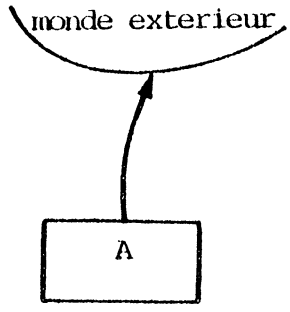


Figure 9-b: Lecture simple

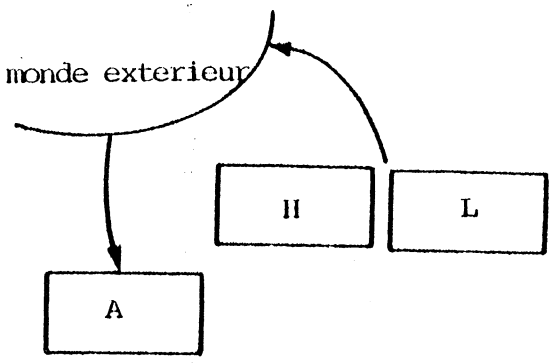


Figure 9-c:Ecriture composée

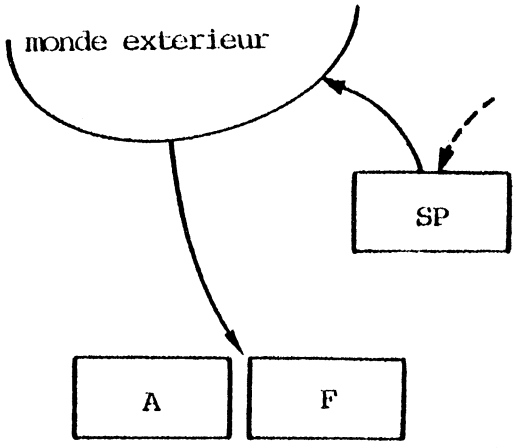


Figure 9-d: Ecriture composée avec effet de bord

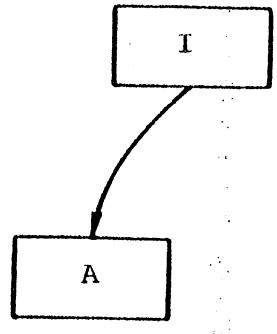


Figure 9-e: Lecture composée du registre I

C -3 Classification des instructions de transfert

Considérons l'ensemble des instructions de transfert affectant un registre R. Comme vu précédemment, une instruction de transfert peut être une instruction de lecture ou d'écriture, simple ou composée, à effet de bord ou non.

On définit ainsi huit groupes d'instructions associés à chaque registre, pouvant être déterminés à l'aide de l'arbre de décision binaire donné en figure 11. La figure 10 donne un exemple d'utilisation de l'arbre pour les instructions étudiées en C-2 .

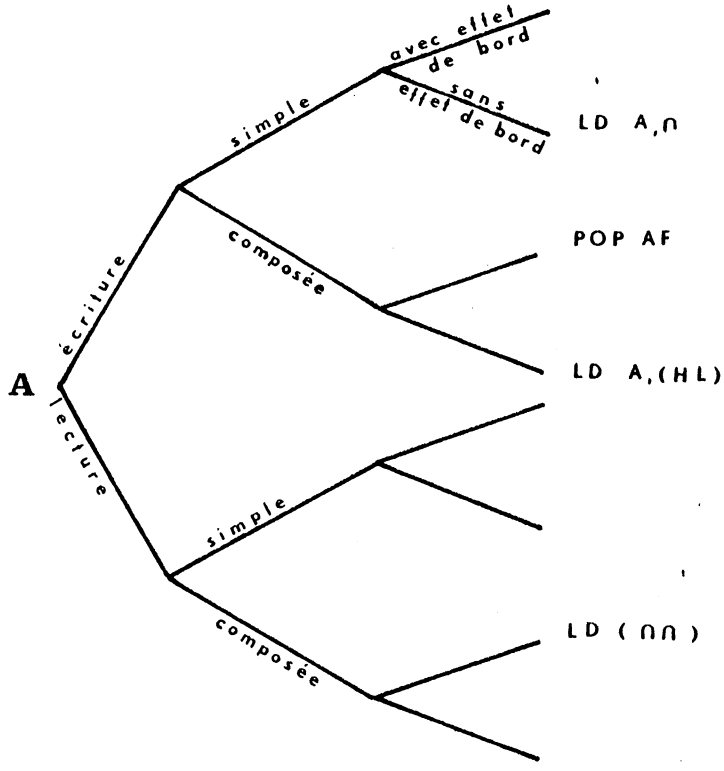


Figure 10 : Utilisation de l'arbre de décision binaire

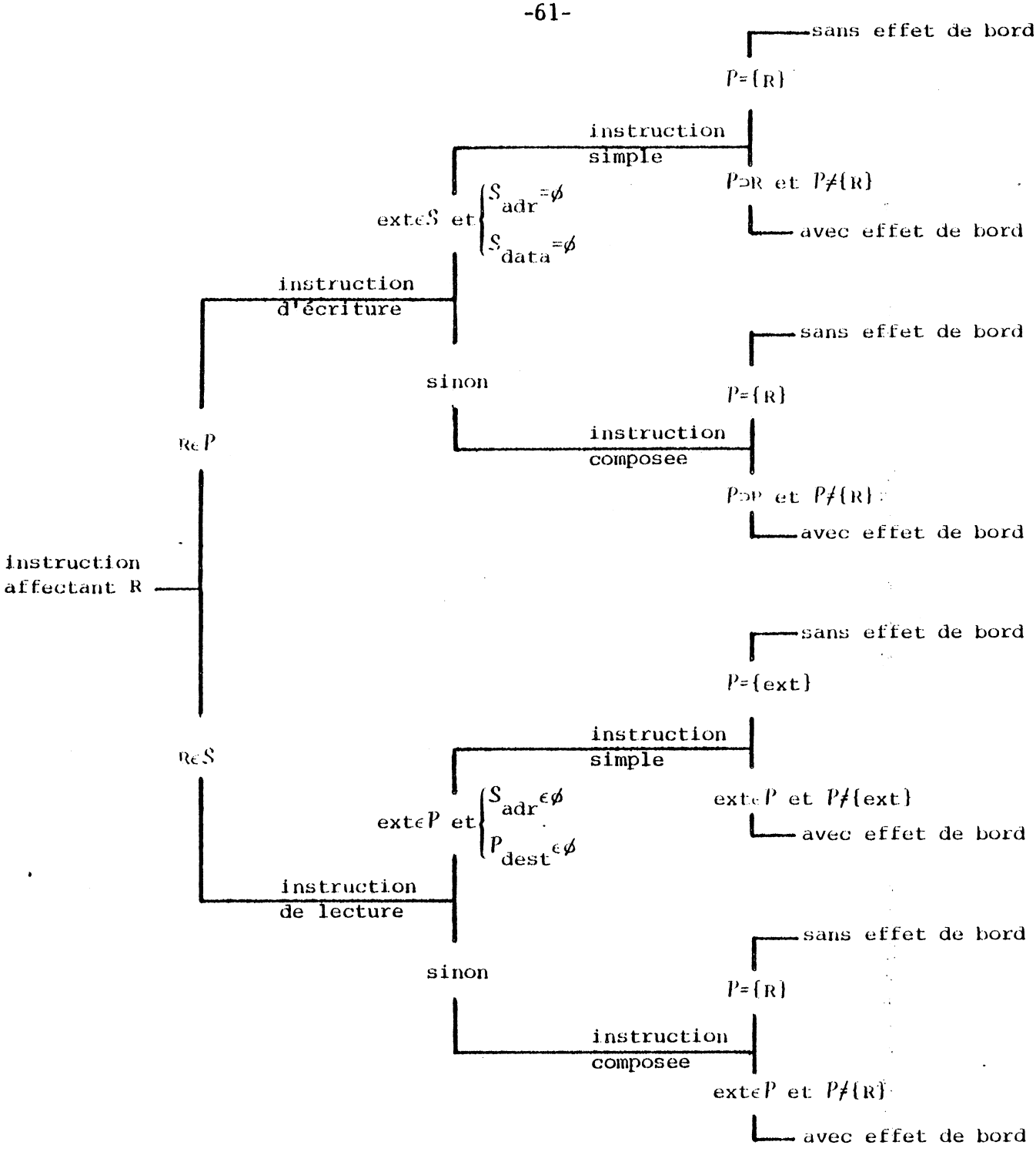


Figure 11: Arbre de décision binaire

Les instructions de transfert affectant chacun des registres seront analysées ; ces instructions étant décrites dans le langage de description défini en IIIC l'analyse à l'aide de l'arbre de décision est aisée. La table 2 présente l'analyse de toutes les instructions de transfert affectant le registre A du Z 80. La table 3 , montre le résultat de l'analyse pour tous les registres du Z 80.

D - SEQUENCES DE COMMANDE ET OBSERVATION ASSOCIEE A UN E M.

On cherche ici les séquences d'instructions permettant de réaliser *la commande et l'observation directe d'un E M.* On suppose que l'analyse des instructions de transfert a été faite en fournissant les tables correspondantes.

D -1 Construction des séquences de commande

Soit E.M l'élément de mémorisation à *commander* :

C : choisir une instruction *d'écriture* pour E M. Deux cas sont possibles :

- Soit il existe au moins une instruction *simple* alors la séquence de commande est réduite à l'une de ces instructions.
- Soit il n'existe que des instructions *composées*. Il faut donc commander les E M qu'elle utilise (S_{adr} et S_{data}), soit itérer l'opération C sur les registres de S_{adr} et S_{data} et faire précéder, à chaque itération, la séquence courante par l'instruction choisie .

Exemple : Instructions de transfert pour l'accumulateur A du Z 80

instruction	description	$S=\{ext, S_{adr}, S_{data}\}$ $P=\{ext, P_{dest}, P_{aux}\}$
POP AF	A ← Mem(SP) F ← Mem(SP-1) SP ← SP-1	{ext, SP , Ø} {-, (A,F), SP} écriture composée avec effet de bord
IN A,(C)	A ← Mem(C)	{ext, C , Ø } {- , A , Ø} écriture composée sans effet de bord
LD A,r	A ← r	{ - , Ø , r} {- , A , Ø} écriture composée sans effet de bord
LD A,I LD A,R	A ← I A ← R	{ - , Ø , I } {- , A , Ø} { - , Ø ,R } {- , A , Ø} écriture composée sans effet de bord

LD A,(HL)
LD A,(BC)
LD A,(DE)
	A ← Mem(HL)
	A ← Mem(BC)
	A ← Mem(DE)
		{ext ,(H,L), Ø} {- , A , Ø}
		{ext ,(B,C), Ø} {- , A , Ø}
		{ext ,(D,E), Ø} {- , A , Ø}
		écriture composée sans effet de bord

LD A,(X+d)	A ← Mem(X+d)	{ext , X , Ø} {- , A , Ø} écriture composée sans effet de bord
IN A,(n) LD A,(nn)	A ← Mem(v_8) A ← Mem(v_{16})	{ext, Ø , Ø} {- , A , Ø} écriture simple sans effet de bord
PUSH AF	Mem(SP) ← A Mem(SP+1) ← F SP ← SP+2	{- , SP ,(A,F)} {ext, Ø , SP} lecture composée avec effet de bord
LD (X+d),A	Mem(X+d) ← A	{- , X , A} {ext, Ø , Ø} lecture composée sans effet de bord

Table 2

LD (HL),A
LD (BC),A
LD (DE),A
	Mem(HL) ← A
	Mem(BC) ← A
	Mem(DE) ← A
		{- ,(H,L), A} {ext, Ø, Ø}
		{- ,(B,C), A} {ext, Ø, Ø}
		{- ,(D,E), A} {ext, Ø, Ø}
		lecture composée sans effet de bord

LD (nn),A OUT (n) ,A	Mem(v_{16}) ← A Mem(v_8) ← A	{- , Ø, A} {ext, Ø, Ø} lecture simple sans effet de bord
OUT (C),A	Mem(C) ← A	{- , C, A} { ext, Ø, Ø} lecture composée sans effet de bord

LD I,A
LD R,A
LD r,A
	I ← A
	R ← A
	r ← A
		{- , Ø , A} {- , I , Ø}
		{- , Ø , A} {- , R , Ø}
		{- , Ø , A} {- , r , Ø}
		lecture composée sans effet de bord

X= { IX , IY } r = { B,C,D,E,H,L }

Suite Table 2 : Analyse des instructions de transfert pour A

	A	r	X	SP	F	I ou R
lecture simple sans e.b	LD A , n LD A ,(nn) IN A ,(n)	LD r,n	LD X,nn LD X,(nn)	LD SP,nn LD SP,(nn)	—	—
lecture simple avec e.b	—	LD dd,nn LD dd,(nn)	—	—	—	—

lecture composée sans e.b sans effet de bord	LD A , r
	LD A , (HL)
	LD A , (BC)
	LD A , (DE)
	LD A , (X+d)
	LD A , I
	LD A , R
	IN A , (C)
		LD r,(X+d)
		LD r,r'
		IN r,(C)
		LD r,(HL)
			—	LD SP,HL
				LD SP,X
					—	—

lecture composée avec e.b sans effet de bord	POP AF	POP dd	POP X	—	POP AF	—
écriture simple sans e.b	LD (nn), A OUT (n) , A	LD (nn),dd	LD (nn),X	LD (nn),SP	—	—

écriture composée sans e.b	LD (X+d),A
	LD (BC), A
	LD (DE), A
	LD I , A
	LD R , A
	OUT (C),A
		LD (HL),r
		LD (X+d),r
		OUT (C), r
			—	—	—	—

écriture composée avec e.b	PUSH AF	PUSH dd	PUSH X	—	PUSH AF	LD A,I
						ou
						LD A,R

X = {IX, IY} r = {B,C,D,E,H,L} dd = {BC,DE,HL}

Table 3 : Instructions de transfert du Z80

Remarque : La séquence commencera donc par une instruction simple, et comportera autant d'instructions que des E.M traversés ou utilisés.

Définition

On appèle *degré de commandabilité* de E M_i le nombre d'éléments de mémorisation traversés lors de la commande de E M_i.

Exemples :

On considère le microprocesseur Z 80 ; l'analyse des instructions de transfert et les tables correspondantes sont données en C -3.

a) Pour le registre IX la séquence de commande est réduite à l'une des instructions simples LD IX,nn ou LD IX,(nn).
Le degré de commandabilité de IX est donc 1.

b) Le registre I ne possède qu'une instruction d'écriture LD I,A.
Cette instruction est composée, le registre A appartient à S_{data}.
Il faut donc commander ce registre. Dans la table on trouve pour le registre A les instructions simples d'écriture IN A,(n);LD A,n et LD A,(nn). Trois séquences de commande seront donc possibles :

- S_1 [IN A(n)
  LD I,A

- S_2 [LD A,n
  LD I,A

- S_3 [LD A,(nn)
  LD I,A

Le degré de commandabilité de I est donc 2.

c) Un cas particulier est celui des registres du banc secondaire. Pour ceux-ci les séquences de commande seront les mêmes que celles de ses homologues du banc principal précédées et suivies de l'instructions de changement de banc de registres.

- Séquence de commande du registre A' $S_{A'}$ [EX / LD A,n / EX]
- Séquence de commande du registre D' $S_{D'}$ [EXX / LD D,n / EXX]

D -2 Construction des séquences d'observation

Soit E M_i l'élément de mémorisation à observer.

O : choisir une instruction de *lecture* de E M_i. Deux cas sont à nouveau possibles :

- Soit il existe au moins une instruction *simple* de lecture, alors la séquence d'observation est réduite à l'une de ces instructions.
- Soit il n'existe que des instructions *composées*. Il faut alors soit commander les E M de S_{adr} par application de l'opération *C*, soit observer les E M de P_{dest} par itération de l'opération *O*. Les nouvelles instructions choisies par *O* suivront la séquence courante.

Définition

On appelle *degré d'observation* d'un élément de mémorisation E M, le nombre d'éléments de mémorisation traversés lors de l'observation de E.M .

Exemple :

a) Pour le registre SP la séquence d'observation est réduite à l'instruction simple de lecture LD (nn),SP.

b) Le registre R ne possède qu'une instruction de lecture LD A,R. Cette instruction est composée P_{dest} = {A}. Il faut donc observer le registre A. Pour cela on trouve les instructions de lecture simples OUT (n),A et LD (nn),A. Les séquences d'observation pour R seront donc :

- S_1 [LD A,R
  LD (nn),A

- S_2 [LD A,R
  OUT (n),A

et son degré d'observabilité égal à 2.

c) Le registre F' appartient au banc secondaire. La séquence d'observation de F' commencera donc par l'instruction de changement de banc de registres EXX; suivra une séquence d'observation du registre F. L'instruction de lecture de F est PUSH AF, S_{adr} = {SP} Il faut alors commander le registre SP. La séquence d'observation pour F' est alors :

[EXX
 LD SP,nn
 PUSH AF
 EXX

et le degré d'observabilité de F' égal à 2.

E - SEQUENCES D'INITIALISATION ET OBSERVATION ASSOCIEES A UN ENSEMBLE D'E M

Soit E M_1 ..... E M_n l'ensemble d'éléments de mémorisation considérés.

Les séquences d'initialisation et observation seront obtenues à partir des séquences définies en paragraphe D pour chacun des E M de l'ensemble considéré. Deux contraintes interviendront dans la construction de ces séquences :

- les degrés de commandabilité et observabilité des E M,
- l'existence ou non d'effet de bord dans les instructions utilisées.

E -1 Séquence d'initialisation pour l'ensemble E M_1.... E M_n

Soient INIT(E M_1),....., INIT(E M_n) les séquences d'initialisation de E M_1, ..., E M_n, et d_c(E M_1), ..., d_c(E M_n), les degrés de commandabilité respectifs.

Une séquence d'initialisation globale est obtenue par concaténation des séquences d'initialisation individuelles avec les contraintes suivantes:

- ordre *décroissant* des degrés de commandabilité,
- si une séquence INIT(E M_i) a des effets de bord sur un (ou des) E M de l'ensemble, par exemple sur E M_j, E M_j sera initialisé après E M_i

Exemple 1 :

Séquence d'initialisation pour les registres A, F, B, C, I, D', SP, E' du Z80.

```
LD  A,V_I        }  d_c = 2     initialisation de I
LD  I,A          }

LD  SP,adr       }  d_c = 2     initialisation de F
POP AF           }              effet de bord sur SP

LD  B,V_B        }
LD  C,V_C        }
EXX              }
LD  D,V_D'       }
EXX              }  d_c = 1     initialisation de B,C,B',A,SP
LD  SP,V_SP      }
EXX              }
LD  E,V_E'       }
EXX              }

LD  A,V_A
```

Remarque 1 : Lors de l'initialisation de F, l'accumulateur A est aussi initalisé ($A \in P_{dest}$). Une séquence plus courte peut être obtenue en supprimant alors la dernière instruction.

Remarque 2 : Les registres de banc secondaire (D',E') peuvent être initialisés consécutivement, en évitant ainsi de répéter les instructions d'échange de banc de registres.

```
EXX
LD  D,V_D'
LD  E,V_E'
EXX
```

Exemple 2 :

Initialisation de tous les E.M du MC 6800

```
LDAA  #V_CC   }
TAP           }  d_c = 2   initialisation de CC

LDAA  #V_A    }
LDAB  #V_B    }
LDX   #V_X    }  d_c = 1   initialisation de A, B, X, S
LDS   #V_S    }
```

Remarque : Toutes les instructions de chargement (sauf TAP) ont un effet de bord sur CC (des indicateurs sont positionnés d'après la valeur transférée). Ces contraintes pour la construction de la séquence globale ne peuvent pas être respectées dans ce cas particulier.

Les solutions qu'on adoptera dans ce cas seront :

- initialiser le registre CC après B, X et SP et initialiser le registre A ensuite pour "minimiser" les effets de bord sur CC.

- lorsque cela sera nécessaire (selon l'instruction testée), initialiser CC toujours à la fin (après B, X et SP) mais ne pas initialiser l'accumulateur A. Ceci permettra d'avoir une valeur de test dans CC qui n'est pas modifiée par la suite de la séquence (cas des branchements conditionnels).

E -2 *Séquence d'observation pour l'ensemble* $EM_1 \ldots\ldots EM_n$

Soient $OBS(EM_1) \ldots\ldots\ldots OBS(EM_n)$ les séquences d'observation de $EM_1 \ldots\ldots EM_n$ et $d_o(EM_1) \ldots\ldots\ldots d_o(EM_n)$ les degrés d'observation respectifs.

Une séquence d'observation globale est obtenue par concaténation des séquences d'observation individuelles avec les contraintes :

- ordre *croissant* des degrés d'observation,
- si une séquence $OBS(EM_i)$ a des effets de bord sur (ou plusieurs) EM de l'ensemble considéré, par exemple sur EM_j ; alors EM_j doit être observé avant EM_i.

Exemple 1 :

Séquence d'observation pour les registres A,B,C,F,I,D',E',SP du Z80.

```
LD  (adr_SP),SP
LD  (adr_A),A
LD  (adr_B),B
LD  (adr_C),C
EXX
LD  (adr_D'),D
LD  (adr_E'),E
EXX
```

d_o =1 observation de SP,A,B,C,D',E'

```
LD    A,I
LD    (adr_I),A
LD    SP,adr_F
PUSH  AF
```

d_o = 2 observation de I et F

Remarque 1 : La simplification pour les registres de banc secondaire a été adoptée.

Remarque 2 : L'observation de F provoque aussi celle de A. La séquence d'observation de A peut être supprimée. Mais, dans ce cas les deux dernières séquences d'observation doivent être interverties pour respecter l'ordre croissant des degrés d'observabilité.

d_o (A) = 1, d_o (F) = 2, d_o (I) = 2

Exemple 2 :

Observation de tous les registres du MC 6800

```
STAA  adr_A
STAB  adr_B
STX   adr_X
STS   adr_S
```

d_o = 1 observation de A,B,X,S

```
TPA
STAA  adr_CC
```

d_o = 2 observation de CC

Remarque : La valeur de CC observée est celle obtenue après les modifications dues à l'exécution des instructions (STAA adr_A, ...). Lorsqu'il sera nécessaire d'observer la valeur de CC sans modifications, sa séquence d'observation doit apparaître en premier. Evidemment le registre A ne sera pas observé dans ce cas (sa valeur est détruite par l'observation de CC).

F - OBSERVATIONS

F-1 Cas des EM qui sont directement commandables et observables mais non pas à travers l'exécution de séquences d'instructions de transfert

Les séquences de commande et d'observation seront déterminées d'après les caractéristiques de tels EM, chacun d'eux étant étudié comme cas particulier.

F -2 Cas des EM qui ne sont pas directement commandables et/ou observables

Les séquences d'initialisation et/ou observation des EM éventuellement nécessaires, ainsi que le (ou les) signal correspondant sont établies .

Exemple :

Bascules IFF1, IFF2 du Z80. Inhibition, validation des interruptions.

Ces bascules sont directement commandables par exécution des instructions:

. EI mise à 1 de IFF1

. DI mise à 0 de IFF1

Leur observation est indirecte. Pour IFF1 il faut observer à travers une interruption masquable.

$\overline{IRQ}$ ⟶

observation { IFF 1 = 0 si $\overline{IRQ}$ est acceptée
IFF 1 = 1 si $\overline{IRQ}$ est inhibée

L'observation d'IFF 2 nécessite le signal $\overline{NMI}$ (interruption non masquable) dont l'activation provoque le transfert de la valeur de IFF 2 dans le bit P/V du registre F. L'observation de cette bascule est obtenue alors par :

$\overline{NMI}$ ⟶

IFF 2 ⟶ F

LD SP,adr$_{IFF\ 2}$
PUSH AF
} observation de F

V - STRUCTURE DES PROGRAMMES DE TEST

Dans ce paragraphe, est presentée la structure du programme de test de conformité,de balayage et des signaux complémentaires.

A - ETAT DU MICROPROCESSEUR

L'ensemble des éléments de mémorisation d'un microprocesseur, noté E_M, définit le nombre d'états possibles. En fait,seul un sous-ensemble, noté E_C, de ces éléments de mémorisation est connu. On définit E_R comme le sous-ensemble de E_C composé des éléments de mémorisation qui sont à la fois directement observables et directement commandables (figure 12).

La valeur des éléments de mémorisation de E_R définit *l'état réduit* du microprocesseur.

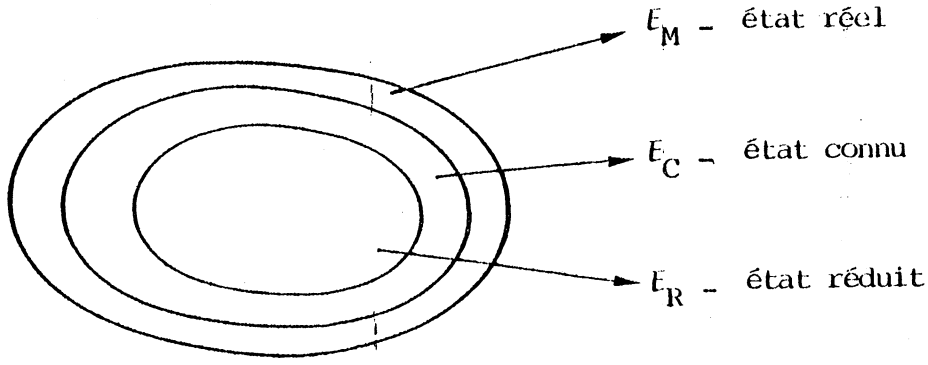


Figure 12: Etat du microprocesseur

B - TEST DE CONFORMITE

B -1 Principe

Le test de conformité vérifie pour chaque instruction, que l'état interne du microprocesseur est modifié correctement et que les valeurs des sorties sont correctes.

La séquence d'instructions pour le test de conformité d'une instruction est composée de :

- l'initialisation de l'état du microprocesseur,
- l'exécution de l'instruction avec les opérandes de test déterminés
- l'observation de l'état du microprocesseur.

Cette séquence est répétée si l'instruction est conditionnelle, pour tester l'instruction dans les différentes situations.

Durant le test de conformité, l'état réduit est systématiquement commandé et observé.

B -2 *Opérandes de test de conformité*

Trois ensembles de valeurs sont associés à chaque instruction (ou type d'instructions) :

$\{X\}_i$: valeurs des sources de l'instruction

$\{Y\}_i$: valeurs des puits internes de l'instruction (avant exécution)

$\{Z\}_i$: valeurs des éléments de E_R non utilisés par l'instruction.

Ces valeurs sont déterminées pour que :

- la valeur des puits internes de l'instruction soit modifiée,
- la valeur des EM non utilisés soit différente de la valeur finale des puits.

La figure 13 illustre le principe du test de conformité.

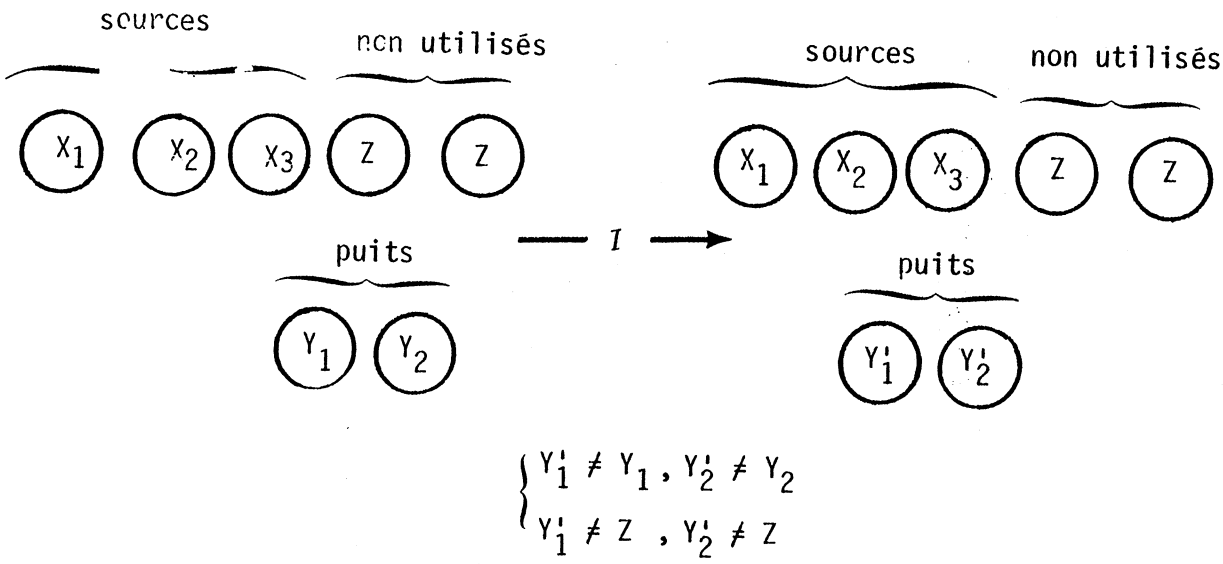


Figure 13: Test de conformité d'une instruction

B -3 *Déroulement du test de conformité d'une instruction*

Avant exécution de l'instruction, il faut que :

- les valeurs des sources externes soient stockées dans la mémoire de test, qui fournira ces valeurs au cours de l'exécution de l'instruction,
- l'état interne du microprocesseur soit initialisé.

Au cours de l'exécution de l'instruction, les valeurs des puits externes seront observés par l'environnement, (ainsi que la valeur du compteur ordinal, et éventuellement les adresses de lecture et/ou d'écriture).

Après exécution de l'instruction, il reste à observer l'état du microprocesseur.

L'initialisation de l'état du microprocesseur est composée de :

- l'initialisation de l'état réduit E_R du microprocesseur,
- l'initialisation des sources ou puits de l'instruction qui sont internes au microprocesseur mais qui n'appartiennent pas à E_R s'il en existe.

La séquence d'instructions et de signaux nécessaire à l'initialisation est construite par concaténation des séquences d'initialisation des EM concernés, dans l'ordre défini en D-1.

L'observation de l'état du microprocesseur comprend:

-l'observation de l'état réduit E_R du microprocesseur,

-l'observation des puits du microprocesseur qui sont internes au microprocesseur mais qui n'appartiennent pas à E_R, s'il en existe.

La séquence d'observation est obtenue, de même, par concaténation des séquences d'observation des EM, dans l'ordre défini en $\mathcal{D}$-2.

Remargue : Le test de conformité est appliqué à chaque instruction. On distingue deux phases dans le test de conformité ; la première concerne toutes les instructions qui n'activent pas d´ EM appartenant à E_R et qui ne nécessitent pas l'activation d'une entrée de type signal.

La deuxième phase concerne les autres instructions, nécessitant l'intervention d'un signal soit pour l'initialisation ou l'observation d'un EM activé, soit pour faire sortir le microprocesseur d'un état d'attente (instruction WAIT, HALT, ...). La deuxième phase sera réalisée au moment du test de signaux.

Exemple : Test de conformité du MC 6800.

Phase 1 : toutes les instructions sauf WAI .

Pour une instruction : I

$$\begin{cases} \text{INIT } (X,Y,Z) \\ I \\ \text{OBS} \end{cases}$$

où INIT et OBS sont respectivement les séquences :

INIT
```
LDAB  #n1
LDX   #m1
LDS   #m2
LDAA  #n2
TAP
LDAA  #n3
```

OBS
```
STAA  m
TPA
STAA  m
STAB  m
STX   m
STS   m
```

Remarques :

- pour l'instruction I, les sources et les puits sont d'abord établies (cf. III C-2).
- les opérandes de test de conformité sont déduits du type d'instruction (opération réalisée,par exemple) . Ils remplacent les paramètres n_i , m_i selon les sources/puits de l'instruction.
- les adresses d'observation des résultats peuvent être quelconques (si l'environnement est réalisé à l'aide de registres de signature).

Exemple 1 : Instruction ABA , d'addition des accumulateurs A et B .

$A \leftarrow A + B \quad S = (A,B)\ ,\ P = (A)\ ,\ v_{add} = (\underset{\text{sources}}{55}\ ,\ \underset{\text{puits}}{AA}\ ,\ \underset{\text{autres}}{00})$

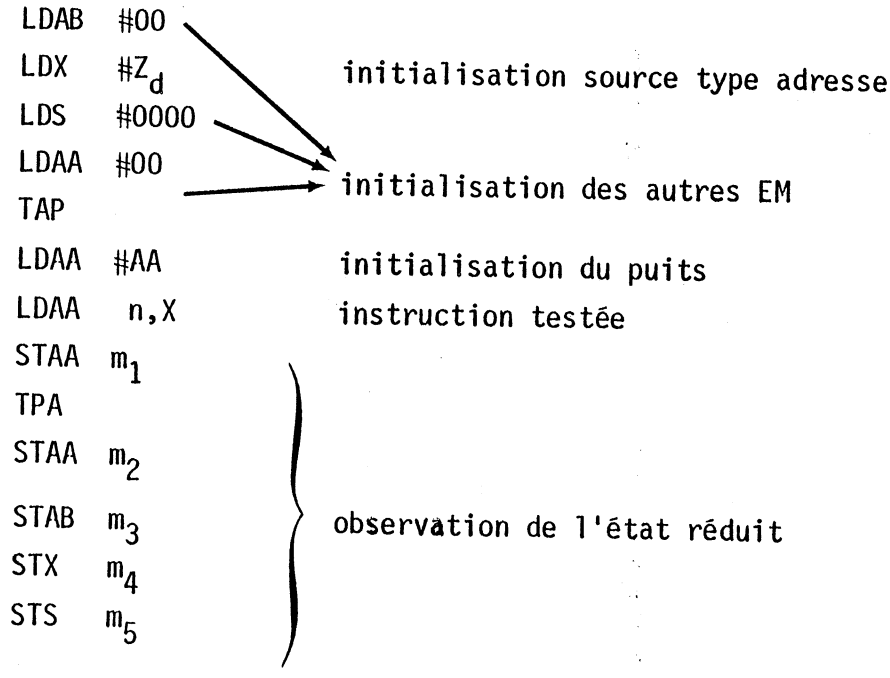
```
LDAB  #55          initialisation source
LDX   #0000        initialisation des autres EM
LDS   #0000
LDAA  #00
TAP
LDAA  #AA          initialisation source
ABA                instruction testée
STAA  m1           observation de l'état réduit
TPA
STAA  m2
STAB  m3
STX   m4
STS   m5
```

Exemple 2 : Instruction LDAA n,X de chargement de l'accumulateur avec adressage indexé.

$A \leftarrow Mem(X + n)$, $S = (X, EXT)$, $P = (A)$

$S = (X, EXT)$, $P = (A)$

$v_{transfert} = (\underset{S_{data}}{55}, \underset{S_{adr}}{Z_d}, \underset{P}{AA}, \underset{reste}{00})$



Remarque : Z_d represente la valeur hexadécimale de l'adresse où se trouve la donnée 55 pour S_{data} .

Phase 2 : instruction WAI sauvegarde de l'état du microprocesseur et attente d'interruption (RESET, $\overline{IRQ}$, $\overline{NMI}$)

Test de conformité : les registres sont initialisés à une valeur donnée puis l'instruction WAI est exécutée. L'observation consiste en l'activation d'un signal $\overline{NMI}$ par exemple avant l'observation de l'état du microprocesseur.

```
INIT  LDAA   #55
      TAP
      LDAB   #55
      LDX    #5555
      LDS    #5555
      LDAA   #55

      WAI

NMI ~>

OBS   STAA   adr_A
      TPA
      STAA   adr_CC
      STAB   adr_B
      STX    adr_X
      STS    adr_S
```

C - LE TEST DE BALAYAGE

C -1 Définition

Le test de balayage consiste à

- partitionner le microprocesseur en blocs,
- associer à chaque bloc un ensemble de vecteurs de test préalablement déterminés,
- envoyer ces données au bloc considéré à travers l'exécution d'instructions adéquates.

Le schéma ci-dessous montre une partition en blocs et pour chacun d'entre eux les types d'instructions qui les activent.

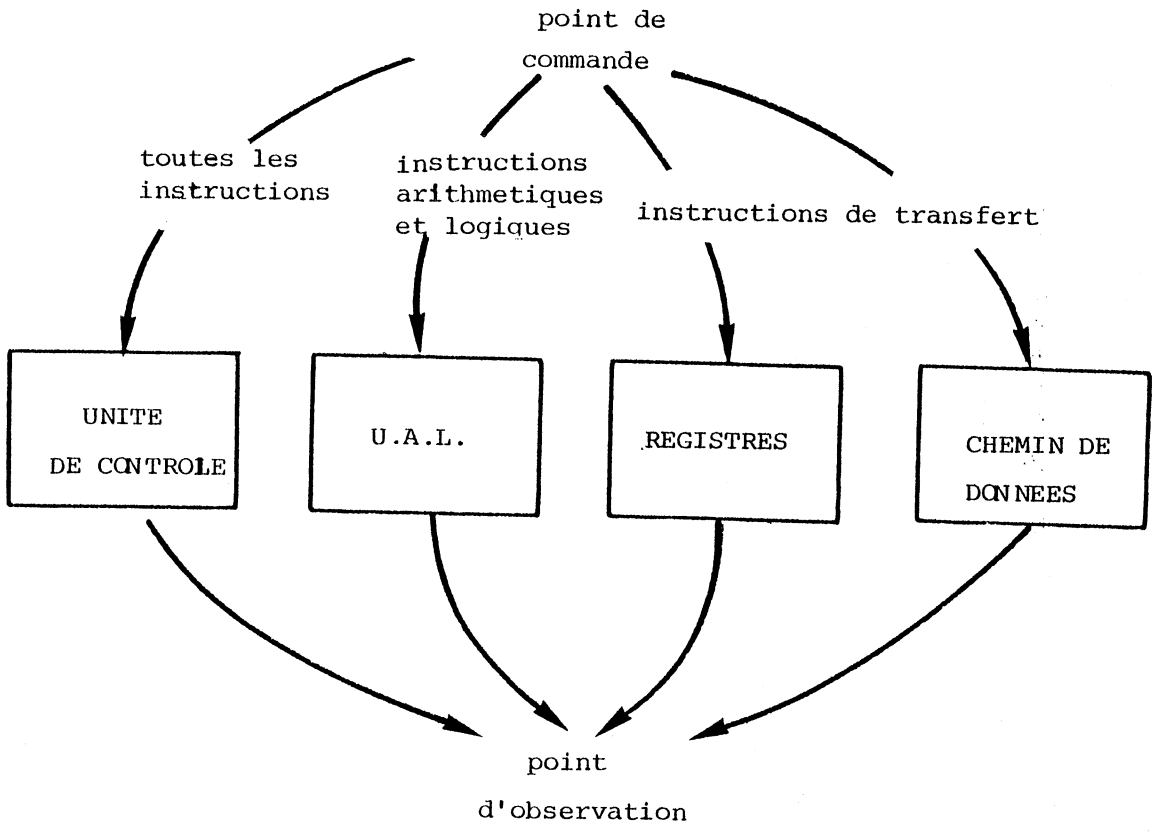


Figure 14 : Partition en blocs pour balayage

L'intérêt de ce test est de pouvoir ajouter ou modifier des modules de balayage de blocs de façon modulaire et flexible suivant les désirs ou les informations de l'utilisateur.

C -2 *Instructions et opérandes pour chaque bloc*

a) Les registres

Les registres peuvent être considérés soit individuellement soit comme un plan mémoire.

- Cas où les registres sont particularisés

Des opérandes de test, déterminés à partir d'hypothèses sur le mauvais fonctionnement d'un élément de mémorisation, sont envoyés à chacun des registres par l'exécution d'une séquence d'instructions de transfert déterminée en fonction du type d'accès :

- si l'accès aux bits est possible, des hypothèses d'interaction entre bits pourront être considérées,
- si les bits d'un registre sont tous accédés en même temps, seules des hypothèses de collages pourront être prises en compte.

La séquence d'instructions correspondant à un élément de mémorisation peut être construite à partir des séquences de commande et d'observation définies dans le paragraphe IV - *D*.

Exemple :

Registres du MC 6800 considérés individuellement.

On suppose que la méthode de test est le WALKING 1/0

Vecteurs de test pour reg_8 = {80, 40, 20, 10, 08, 04, 02, 01}

Vecteurs de test pour reg_{16}= {8000, 4000, ..., 0001}

reg_A

```
LDAA  #80
STAA   adr
LDAA  #40
STAA   adr
  .
  .
  .
LDAA  #01
STAA   adr
```

reg_X

```
LDX   #8000
STX    adr
LDX   #4000
STX    adr
  .
  .
  .
LDX   #0001
STX    adr
```

reg_{CC}

```
LDAA  #80
TAP
TPA
STAA   adr
  .
  .
  .
LDAA  #01
TAP
TPA
STAA   adr
```

En [Vel 81], une séquence minimale a été définie ainsi que la manière dont les opérandes doivent être obtenus pour activer tous les éléments de mémorisation d'un microprocesseur quelconque.

- Cas où les registres sont banalisés (test mémoire)

  L'ensemble des registres est considéré comme une mémoire RAM (organisée généralement en mots de 8 bits). Des méthodes de test classiques de mémoire (WALKING, MARCHING, GALLOPING, ...) seront donc appliquées.

Ces méthodes sont définies, en général, comme une séquence d'activations (lecture/écriture) des cellules de la mémoire, considérée comme une matrice de bits.

Dans notre cas, leur application consiste essentiellement [Ngu 81]:

- à adapter ces méthodes pour les appliquer à une mémoire organisée en mots,
- à écrire la séquence d'instructions correspondante [Vel 81][Ngu 81]

L'intérêt d'utiliser une méthode de test de type mémoire est :

- une bonne couverture de pannes,
- une estimation facile de la complexité.

b) UAL

Trois approches sont étudiées :

- Test avec opérandes aléatoires

  Dans ce cas, une analyse structurelle du circuit ou de la portion de circuit activée par la fonction (instruction par exemple) permet d'évaluer le nombre d'opérandes aléatoires à générer. Cette analyse est fondée sur la recherche de la panne la plus difficile à détecter [The80]. On retrouve alors tous les problèmes de l'approche structurelle (niveau de représentation, modélisation de la panne).

  On peut se servir alors des résultats (sous forme d'abaques) donnés en [The 78] : il s'agit d'associer à une addition de n bits une séquence aléatoire de longueur L donnée dans ces abaques et correspondant à la complexité d'un additionneur n bits.

- Test avec opérandes exhaustifs

Cela semble la seule approche rigoureuse étant donné la méconnaissance de la structure réelle, mais elle n'est pas applicable en pratique (2^n vecteurs de test pour n bits en entrée).

- Approche déterminste

Pour les opérations décrites par un ensemble d'équations booléennes, une analyse par bit ou tranche de bits permet de déterminer des opérandes "significatifs" pour ces bits, puis de les étendre par symétrie ou répétitivité sur l'ensemble des mots. Dans l'Annexe 1 la méthode d'obtention de ces opérandes est développée.

Pour les fonctions plus complexes (multiplication, division,...), on peut se ramener à des descriptions algorithmiques [Kai 79] et chercher des opérandes réalisant un test algorithmique.

c) Adressage

Il s'agit de tester le matériel impliqué dans les différents modes d'adressage, dans le cas où l'on suppose (ou on sait) que ce matériel est spécifique.

Il faudra tester :

-l'incrémentation des registres adresse (pointeur de pile, registres index, compteur programme),

-l'additionneur d'adresses (adressage relatif, adressage indexé),

-le bus adresse.

Les opérandes sont ceux définis pour les registres ou opérateurs de traitement mais ils seront amenés par des instructions activant les registres ou opérateurs d'adressage.

D - LE TEST DES SIGNAUX COMPLEMENTAIRES

D-1 Cette étape de test concerne les signaux et les instructions associées.

Parmi les instructions on trouve :

- les instructions activant des EM indirectement commandables et/ou observables, ces instructions sont testées par un test de type conformité. Les séquences d'initialisation et d'observation comporteront l'initialisation et l'observation de l'état réduit E_R et des EM supplémentaires activés par l'instruction.
- les instructions mettant le microprocesseur dans un état d'attente dont il ne peut sortir que grâce à l'intervention d'un signal ; ces instructions sont testées en conformité.

Parmi les signaux de type sortie, on peut distinguer :

- les signaux de sortie activables par le jeu d'instructions,
- les signaux de sortie qui ne sont activés que par un signal de type entrée.

La non activation intempestive de ces signaux a été testée tout au long des précédents tests, leur activation sera vérifiée au cours du test des signaux.

Les signaux de type entrée peuvent se classer en :

- signaux de type interruptif, ayant des conséquences fonctionnelles (IT, RESET , ...) ; ces signaux forcent le microprocesseur à effectuer un branchement (après sauvegarde éventuellement). Ces signaux seront testés à travers leurs conséquences fonctionnelles.

- signaux de type suspensif, qui suspendent l'exécution des instructions; ces signaux seront testés en vérifiant la non activation des sorties du microprocesseur.

- autres signaux, qui sont en général de type dialogue, et qui ont pour effet d'activer un signal de sortie.

Tous ces signaux seront testés dans toutes les configurations significatives (tous les états du microprocesseur ayant une conséquence sur la prise en compte ou l'émission des signaux).

La mise en oeuvre du test des signaux est très liée au testeur utilisé.

D-2 Exemple : Le test des signaux interruptifs

a) Description du système d'interruption en vue du test

Le système est présenté sous forme d'un arbre où sont portées les informations concernant chacune des interruptions. Ces informations obtenues à partir du manuel d'utilisation sont la liste des signaux d'interruption, leurs conditions d'acceptation et leurs effets sur l'état du microprocesseur.

Liste des signaux d'interruption

* On peut distinguer plusieurs cas, par exemple :

- une ligne par type d'interruption

  RESET

  INTERRUPTION NON MASQUABLE

  INTERRUPTION MASQUABLE

- une ligne unique avec un code de priorité.

A chacune des interruptions ou niveaux de priorité correspondra une branche dans l'arbre de description.

* Pour chaque interruption sont décrites *les conditions d'acceptation et d'inhibition*, c'est à dire l'état du matériel spécifique tel que bascules ou registres de masques.

  Une interruption de type non masquable est toujours acceptée ; il lui sera associée une branche simple dans l'arbre.

  A une interruption de type masquable seront associées deux branches (interruption validée, interruption inhibée).

* Action de l'interruption sur l'état du microprocesseur

. sauvegarde de l'état du microprocesseur, qui peut être interne ou externe, totale ou partielle (elle concerne au minimum le compteur programme).

. branchement à l'adresse de traitement. L'élaboration de cette adresse diffère selon les microprocesseurs. A chaque interruption peut être associé un ou plusieurs modes d'adressage soit une ou plusieurs branches dans l'arbre.

. modifications de l'état du microprocesseur.

L'arbre de description présente un double intérêt : d'une part sa généralité et adaptabilité aux systèmes étudiés, d'autre part sa capacité à servir de support à la génération automatique du programme de test.

On donne en figures 15 et 16 la description correspondant aux systèmes d'interruption des microprocesseurs MC 6800 et Z80.

b) Module de test associé à une interruption

A une interruption on associe un module de test comportant trois parties :

- Initialisation
- Attente d'interruption
- Observation.

* Initialisation

On définit une séquence d'instructions, séquence d'initialisation, dont l'exécution par le microprocesseur sous test provoque :

- le positionnement du matériel spécifique à la validation de l'IT et au branchement vers le programme de traitement correspondant (registres utilisés pour le calcul d'adresse de traitement).

- le chargement de l'état du microprocesseur et de la zone de sauvegarde à des valeurs initiales connues.

La séquence d'initialisation est la séquence d'initialisation de l'état éventuellement complétée par l'initialisation des éléments de mémorisation spécifiques au système d'interruptions.

* Attente de l'interruption

Pour certains microprocesseurs il est difficile de réaliser un montage matériel assurant que le signal d'IT survienne et soit pris en compte à un instant déterminé. Le signal d'IT sera considéré comme asynchrone ; le microprocesseur sous test doit donc attendre l'IT. Trois solutions ont été envisagées :

- Attente de l'interruption par exécution d'une boucle infinie ; dont on sort par l'IT elle-même, cette solution ne permet pas de tester le système d'inhibition des IT.
- Attente de l'IT par exécution d'une boucle contrôlée par un compteur.
- Attente de l'IT par l'exécution d'une série d'instructions identiques (des NOP par exemple).

La première solution étant écartée, les deux dernières solutions imposent un retour "naturel" après traitement de l'IT.

En effet, le signal d'IT survient de façon asynchrone et l'état du microprocesseur (en particulier son compteur programme) à la prise en compte de l'IT n'est pas toujours le même.

Le temps d'exécution de la boucle contrôlée ou de la séquence d'instruction est déterminé pour chaque microprocesseur en fonction des contraintes matérielles dues au banc de test.

* Observation

La vérification du bon déroulement des actions correspondant à la prise en compte (ou à l'absence de prise en compte) d'une IT passe par l'exécution d'une séquence d'observation dont les fonctions sont de :

- vérifier le branchement à l'adresse de traitement,
- vérifier l'état après prise en compte de l'IT,
- vérifier la sauvegarde et la restauration de l'état avant IT.

Que l'on cherche à vérifier l'inhibition d'une IT ou sa prise en compte, la séquence d'observation sera la même mais doit fournir des résultats différents.

Construction de la séquence d'observation

- La vérification du branchement à l'adresse de traitement est faite implicitement,
- On place à l'adresse de traitement de l'IT une séquence d'observation de l'état du microprocesseur. Cela permet de vérifier que la prise en compte de l'IT a modifié ce qu'il faut (masquage des autres IT en particulier).
- Puis la valeur des registres qui ont été sauvegardés (autres que PC) est modifiée ; la séquence est terminée par un retour de l'IT, soit une instruction spécialisée (RTI par exemple) soit en la simulant.

- A la suite de la boucle ou séquence d'attente de l'interruption est placée une séquence d'observation de l'état du microprocesseur

La figure 17 illustre la structure du programme de test d'une interruption.

c) Structure globale du test du système d'IT

Chaque interruption sera testée dans toutes ses configurations possibles soit acceptée inhibée (si c'est possible) et différents modes d'adressage s'ils existent.

Donc, toutes les branches de l'arbre de description du système considéré seront parcourues.

A chaque parcours correspondra un module de test (initialisation, attente, observation), l'ensemble de tous les modules constitue le programme de test du système d'IT.

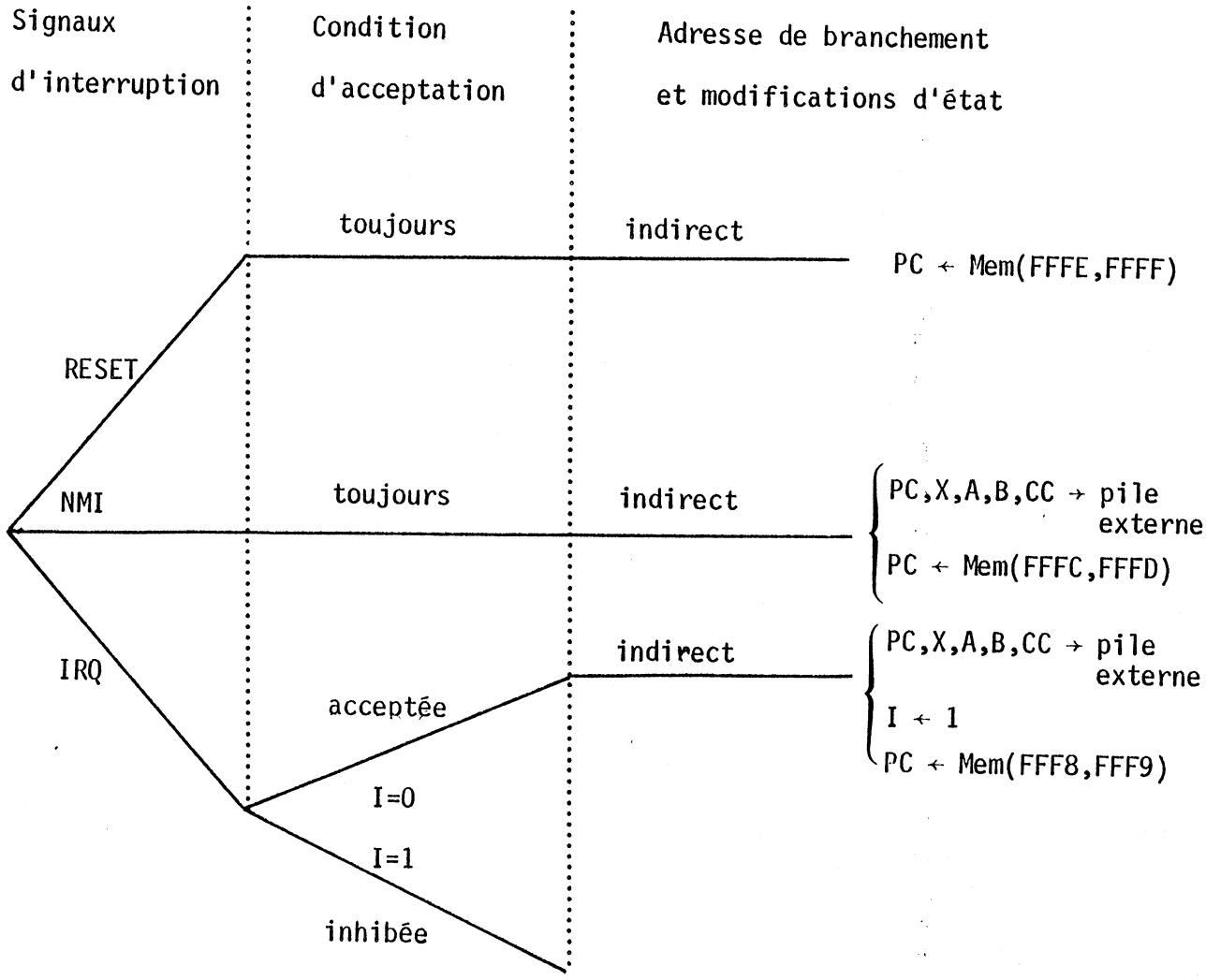


Figure 15 : Signaux interruptifs du MC6800

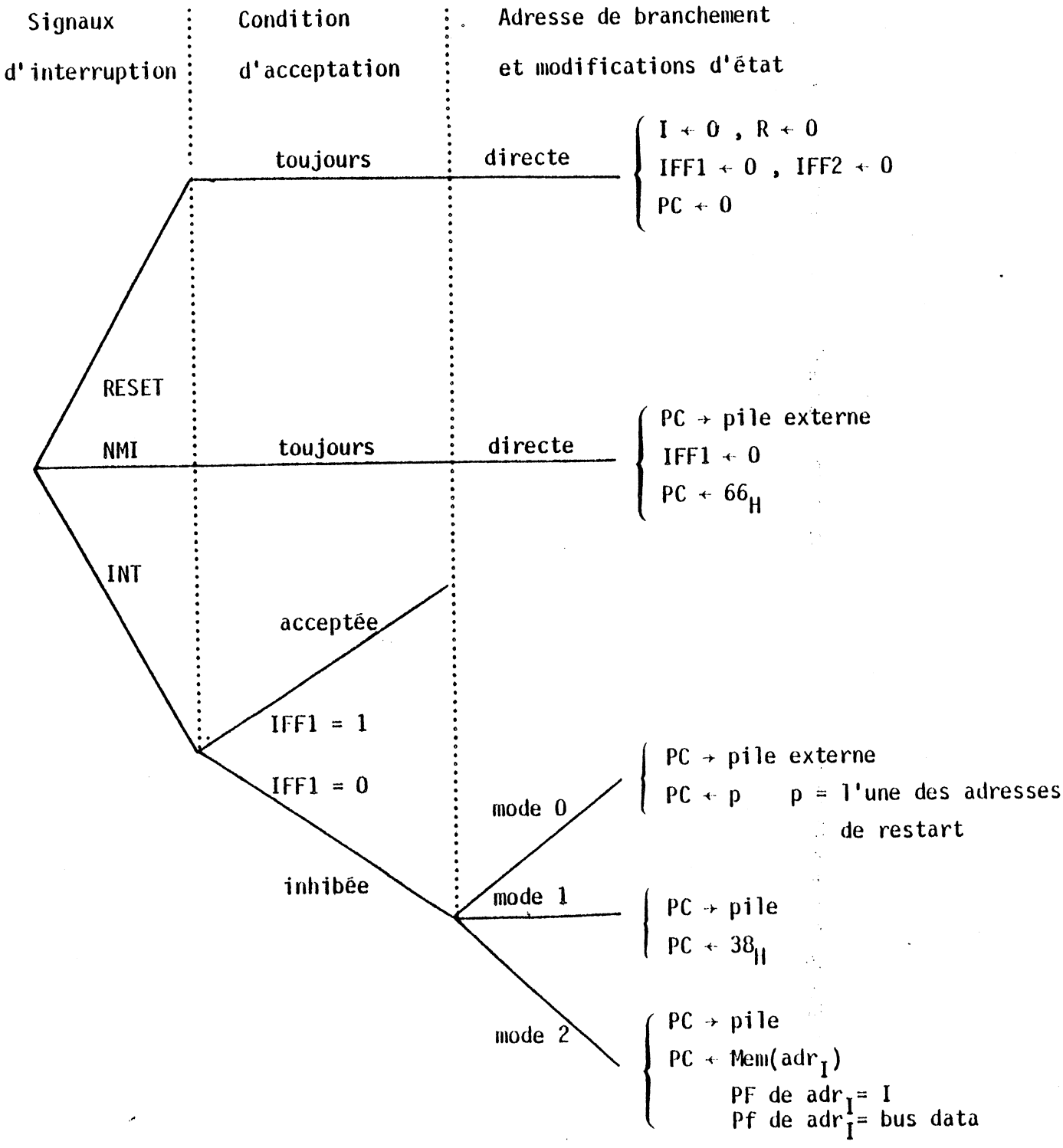


Figure 16 : Signaux interruptifs du Z80

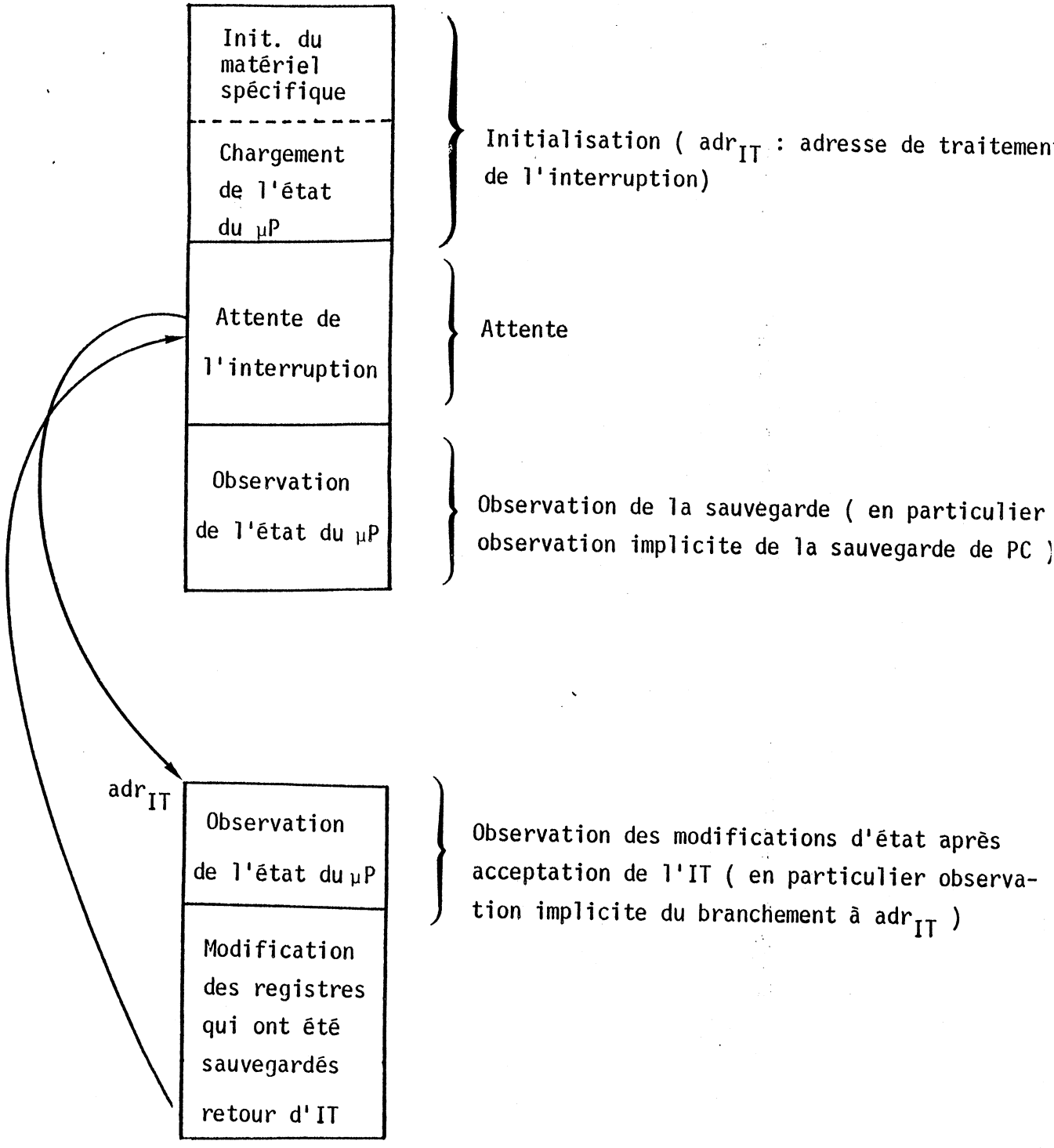


Figure 17 : Test d'une interruption

VI - GÉNÉRATION AUTOMATIQUE DES PROGRAMMES DE TEST

A - *INTRODUCTION*

Le test de circuits (V)LSI requiert la génération d'un nombre élevé de vecteurs de test. Ce nombre pouvant varier entre quelques milliers et quelques dizaines ou centaines de milliers de vecteurs, selon la complexité du circuit testé, leur génération manuelle devient difficilement envisageable.

La génération automatique de vecteurs de test, dite aussi algorithmique, exploite le caractère itératif des programmes de test construits à partir des méthodes comportementales.

Trois approches ont été relevées dans la bibliographie.

a) L'utilisation de macroassembleurs comme outils d'aide à l'écriture de programmes de test est proposée en [Pat 80] . Des "macros" spécifiques, dont l'utilisation simplifie l'écriture des programmes de test sont définies. On rappelle qu'une macro est une séquence d'instructions paramétrable qui est insérée dans le programme au moment de l'assemblage. Ceci permettra une écriture plus rapide des programmes mais entrainera des difficultés dans leur mise au point. En effet, l'utilisation des macros est complexe, pour certaines primitives typiques du test (itération sur des ensembles par exemple), car il faut imbriquer les appels de macros.

b) En [Pas 80] les possibilités offertes par le langage PASCAL, sont étudiées. Ce langage apparait comme très adapté aux problèmes de l'écriture d'algorithmes de génération de vecteurs de test. Les avantages significatifs de ce langage sont en effet :

- sa nomenclature,
- sa structure de blocs,
- sa diversité de types de données (ensembles, tableaux, ...),
- la variété de primitives de contrôle du flot de données, IF...THEN... ELSE , WHILE...DO..,REPEAT...UNTIL, FOR... DO, CASE...OF, ...

Le langage PASCAL est déjà fourni comme langage de test dans certains équipements de test (Megatest, Teradyne, Watking-Johnson). La Société Megatest propose une extension du PASCAL, le langage B-PASCAL spécialement orienté vers les problèmes de la génération de vecteurs de test pour LSI.

c) Un langage haut-niveau spécifique aux problèmes du test a été développé par IBM. Ce langage, "PLT" [Kiz 80] résulte d'une étude des langages utilisés par le département de test d'IBM, et en réunit leurs meilleures caractéristiques pour le test. Il permet à l'utilisateur de décrire, en termes généraux, le dispositif sous test, ainsi que les tests à appliquer. En particulier PLT a été utilisé pour la génération de tests fonctionnels des circuits tels que les RAM et les microprocesseurs.

On a abordé le problème de l'écriture de programmes de test pour microprocesseurs, en deux étapes :

- définition et réalisation *d'outils d'aide à l'écriture de programmes de test.* Il s'agit du système ROBIN présenté dans la section II. Le système propose un langage permettant l'écriture des primitives nécessaires à la génération des programmes de test (spécification d'ensembles, primitives d'itération sur des ensembles, appel de macros, ...).

  Le système ROBIN a été utilisé pour l'obtention des programmes de test pour les microprocesseurs 8 bits courants (MC 6800,Z80,2650).

- Spécification d'un *générateur automatique de programme de test* pour microprocesseurs appelé GAPT. Ce système doit, à partir de la description d'un microprocesseur, générer les programmes de test d'après la méthode définie dans les paragraphes précédents. La phase de spécification du générateur GAPT est terminée, l'étude a été menée en collaboration avec la Société EMD ; le système doit être opérationnel début 83. L'étude des principales caractéristiques de GAPT fait l'objet de ce paragraphe.

B - *LE SYSTEME GAPT* [VEL 82b]

B -1 *Principe*

A partir de la description du microprocesseur et des opérandes de test prédéterminés, ce système génère les modules du programme de test en langage Assembleur du microprocesseur testé (figure 18).

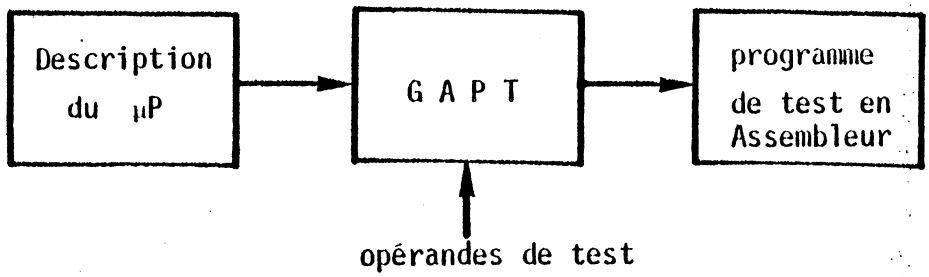


Figure 18

Nous avons choisi de générer des programmes en assembleur ; ce choix, bien qu'imposant l'assemblage du programme généré se justifie par les arguments suivants :

- La génération en assembleur permet une *description compacte* du jeu d'instructions, facilitant ainsi la tâche de description du microprocesseur.

- La *lisibilité* des programmes de test générés facilite leur mise au point et leur modifications éventuelles.

- La *portabilité* des programmes de test fournis. En effet, l'assemblage pourra être réalisé par l'utilisateur, en assurant ainsi la compatibilité avec ses propres systèmes de développement. Il est à observer que l'utilisateur de microprocesseurs, dispose toujours des assembleurs correspondants.

Le synoptique donné en figure 19 représente la manière dont les programmes de test sont obtenus ainsi qu'une représentation globale du générateur.

B -2 *Le langage de description*

Une des contraintes principales du langage de description est son universalité : tout microprocesseur doit pouvoir être décrit.

Le jeu d'instructions des microprocesseurs 16 bits (et de certains microprocesseurs 8 bits) est décrit de manière compacte dans les manuels d'utilisation : pour chaque type d'instruction est donnée une description comportant l'ensemble des modes d'adressages et/ou de registres valides, ainsi que la taille des données sur lesquelles l'instruction peut travailler.

Le langage de description défini en II a été complété par les extensions nécessaires à la description de groupes de registres, des modes d'adressage, et d'ensembles de modes d'adressage.

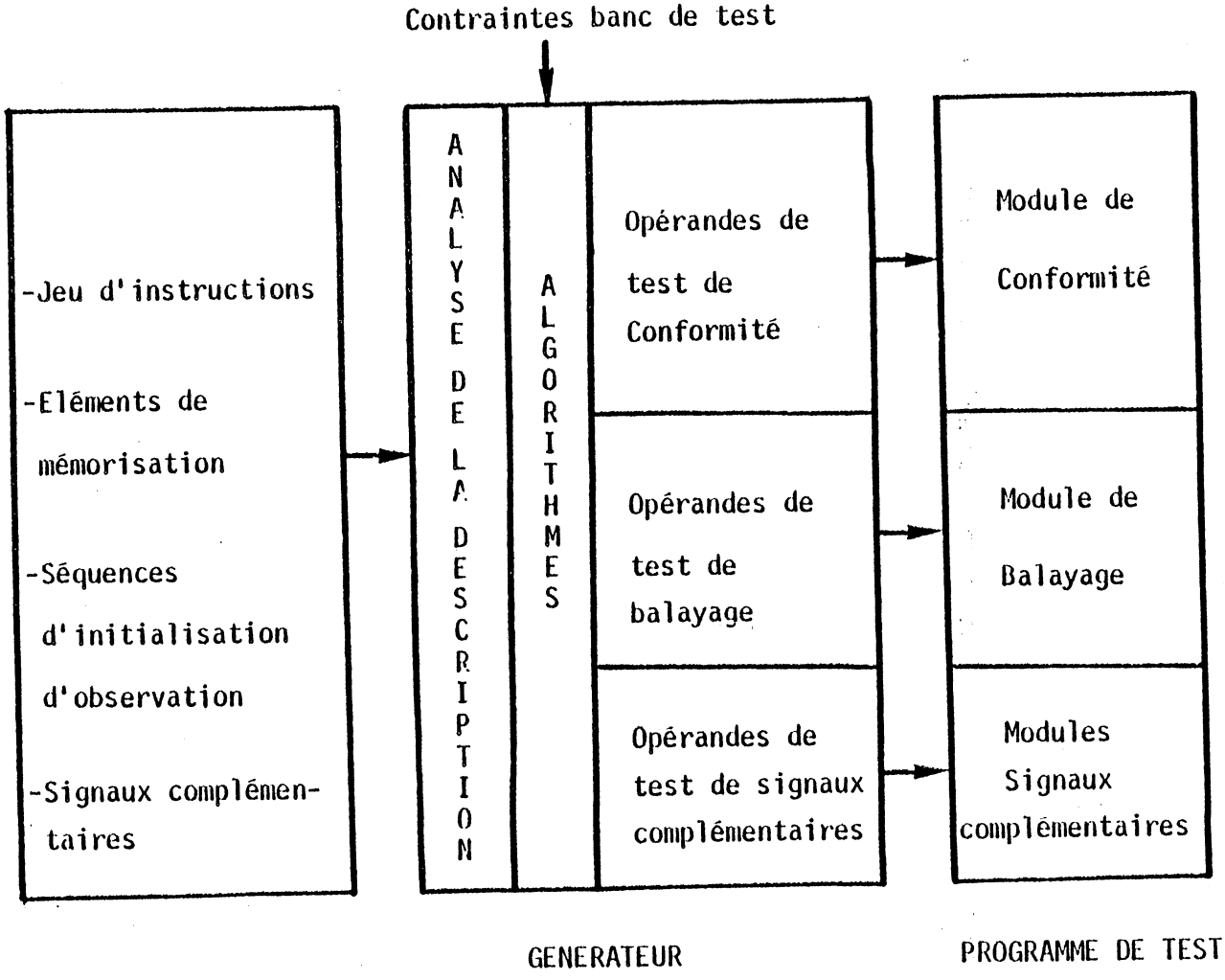


Figure 19 : Synoptique du générateur GAPT

La description d'un microprocesseur se présente comme un ensemble d'unités regroupées en deux parties : partie déclaration et partie fonctionnement.

a) Déclaration

Dans cette partie sont donnés les renseignements matériels qui seront exploités pour le test. Ce sont :

- les éléments de mémorisation,
- les opérateurs,
- les signaux.

1) Section éléments de mémorisation internes

Chaque EM est décrit par :

- son nom,
- sa taille en nombre de bits,
- une séquence d'initialisation paramétrée,
- une séquence d'observation paramétrée.

2) Section groupes de registres

Cette section définit les différents groupes de registres qui ont des caractéristiques communes, par exemple pouvant être associées au même code mnémonique.

3) Section organisation des registres

Des éléments de mémorisation de taille diverse peuvent être définis à partir de l'association des registres, ou de la dissociation d'un registre en plusieurs éléments de mémorisation de taille inférieure. Dans cette section sont spécifiées ces différentes organisations des registres.

4) Section modes d'adressage

Dans cette section sont répertoriés les modes d'adressage. Leur description comporte deux parties : l'une décrivant la syntaxe assembleur des instructions utilisant ce mode d'adressage, l'autre décrivant les EM ou ensembles de EM intervenant dans le calcul de l'adresse.

5) Les opérateurs

Liste d'opérateurs :

- ceux correspondant aux différentes fonctions de l'UAL,
- des opérateurs spéciaux.

6) Les signaux

Les signaux d'entrée-sortie sont décrits à l'aide des noms symboliques éventuellement groupés (cas des bus adresse et données).

b) Fonctionnement

Cette partie décrit les fonctionnements possibles du microprocesseur à travers la description des activations "apparentes" lors de l'exécution des instructions, ou l'arrivée de signaux d'entrée ayant des conséquences fonctionnelles.

1) Instructions

Cette section contient la description de *toutes* les instructions. Une description compacte paramétrée est donnée pour l'ensemble d'instructions qui ont le même code mnémonique. Une telle description compacte sera appellée PSEUDO INSTRUCTION. Elle comporte :

- le code mnémonique, avec la spécification des parties variables (correspondant à des ensembles de registres, des modes d'adressage, et la taille de données manipulées dans le cas des microprocesseurs permettant de manipuler des données 8 16 et 32 bits par exemple).

- La description des transferts entre registres et des opérateurs activés par l'instruction (ou le groupe d'instructions), telle qu'elle est présentée en II.

2) Signaux

Les signaux d'entrée ayant des conséquences fonctionnelles sont décrites par l'ensemble d'activitions apparentes.

B -3 *Architecture du générateur*

L'architecture globale du générateur a été donnée dans le synoptique de la figure 19.

Le générateur produira soit un programme de test de conformité, soit un programme de test de balayage, en utilisant les opérandes de test adéquats.

La figure 20 donne la manière dont sera réalisé le générateur et les différentes parties le constituant.

Le coeur du générateur automatique est le bloc analyseur. Celui-ci lira la description du microprocesseur et analysera son contenu à l'aide de ses tables d'analyse lexicale et syntaxique. Le rôle de l'analyse lexicale est de reconnaître les unités lexicales du langage (mots clés réservés, les identificateurs, les valeurs, des symboles spéciaux ...). Ces unités lexicales sont ensuite traitées par l'analyse syntaxique qui vérifiera la syntaxe du langage et déclenchera au besoin les actions sémantiques.

Les actions sémantiques seront un ensemble de sous programmes qui permettront de ranger toutes les informations données dans des tables et pour chaque instruction décrite de générer le programme de test correspondant en utilisant les opérandes de test adéquat

Certains blocs du générateur seront directement obtenus à partir d'outils logiciels existant à l'IMAG. Notamment, un transformateur de grammaires [Del 80] permet d'obtenir des tables d'analyse syntaxique partir d'une définiton de la grammaire d'un langage. L'analyseur de ces tables est également disponible.

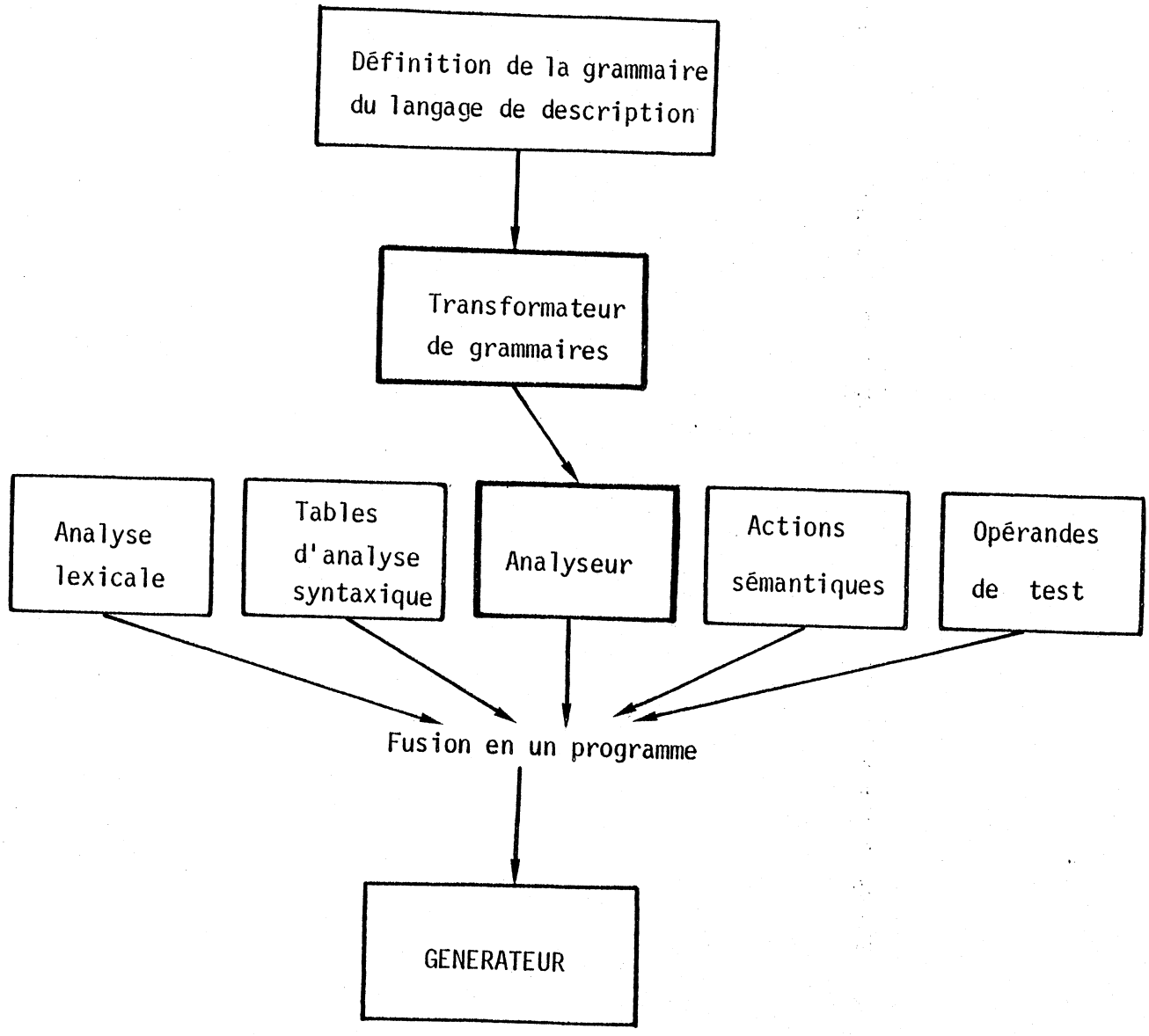


Figure 20 : "Architecture" logicielle du générateur

La réalisation du générateur consistera alors à définir :

- la grammaire du langage de description,
- l'analyse lexicale,
- les opérandes de test,
- les actions sémantiques nécessaires à la génération des programmes de test.

Le langage de description ainsi que les algorithmes de génération sont présentés en [Vel 82 a].

L'exemple suivant présente une description partielle du MC68000 .

Exemple :

Microprocesseur 68000 de Motorola

MC 68000

Taille-mot-mémoire : 8

Etat

registre d'état SR : (T,*,S,*,*,I2,I1,I0,*,*,*,X,N,Z,V,C)

compteur-programme PC : 32

registres 32 : (D0,D1,D2,D3,D4,D5,D6,D7,A0,A1,A2,A3,A4,A5,A6,A7)

Dissociations

SR :(:8, CCR : 8)

Groupes-registres

Dn : (D0,D1,D2,D3,D4,D5,D6,D7)

An : (A0,A1,A2,A3,A4,A5,A6)

Dx : Dn

Dy : Dn

Rn : Dn ∪ An

Notations

Vi : (V8 : 8, V16 : 16, V32 : 32)

ADR : (Ad8 : 8, Ad16 : 16, Ad32 : 32)

DEPL : (d_8 : 8 , d_{16} : 16)

Format

SIZE : (B : 8, W : 16, L : 32)

FIN

Exemple :

MC 68000

MODE D ADRESSAGE : (ea,src ,dst)

DR : <Dx> / Dx ; + 0

AR : <Ax> / Ax ; + 0

ARI : (<Ax>) / MEM * size (Ax) ; + 0

ARIPI : (<Ax>) + / MEM * size (Ax) ; + 0

ARIPD : - (<Ax>) / MEM * size (Ax - size) ; + 0

ARID : <d16> (<Ax>) / MEM * size (Ax + d16) ; + 2

ARIX : <d8> (<Ax>,<Rn>) / MEM * size (Ax + Rn + d8) ; + 2

ASA : <od16> / MEM * size (Ad16) ; + 2

ALA : <od32> / MEM * size (Ad32) ; + 4

PCD : <d16> (PC) / MEM * size (PC + d16) ; + 2

PCX : <d8> (PC,<Rn>) / MEM * size (PC + Rn + d8) ; + 2

IMM : <Vi> / Vi ; + size

GROUPE MODES

data : Tous SAUF (AR)

memory : data SAUF (DR)

control : memory SAUF (ARIPI, ARID, IMM)

aberable : tous SAUF (PCD, PCX, IMM)

ma : memory SAUF (PCD, PCX, IMM)

da : data SAUF (PCD, PCX, IMM)

ca : control SAUF (PCD, PCX, IMM)

FIN

Exemple :

GEN

ADD . <size> <ea>,<Dn> : Dn ⟵ Dn + ea

ea : tous

si SIZE = B alors ea : tous SAUF (ARI)

FIN-GEN

GEN

MOVE . <size> <SRC>,<dst> : dst ⟵ SRC

[size : (B,W,L)]

SRC : tous

si SIZE = B alors SRC : tous SAUF (ARI)

dst : tous

FIN-GEN

SECTION II

REALISATION EXPERIMENTALE

INTRODUCTION

Dans cette section sont présentés les résultats d'une étude de faisabilité préliminaire ayant permis de valider l'approche comportementale du test de microprocesseurs et d'aborder le problème de la génération automatique.

Dans cette première étude :

- des programmes de test comportemental ont été écrits pour certains microprocesseurs permettant ainsi d'obtenir des résultats pratiques sur cette approche.
- Un outil logiciel d'aide à l'écriture des programmes de test a été réalisé (logiciel "ROBIN").
- Un testeur a été construit, permettant d'expérimenter rapidement l'efficacité de cette approche.

I - ECRITURE DES PROGRAMMES DE TEST

A - *DESCRIPTION DES MICROPROCESSEURS [Vel 80a][Vel 80b]*

Dans cette première approche une description graphique des instructions du microprocesseur a été utilisée, à savoir le *graphe d'exécution abstraite* ; cette première description nous a paru simple, facile à manier et très intuitive. Il s'agissait de définir les informations minimales nécessaires au test; cela nous a permis de mieux aborder dans un second temps l'élaboration d'un langage de description plus formel et plus adapté à la génération automatique.

A -1 *Graphe d'exécution abstraite*

On associe à chaque instruction du microprocesseur un *graphe d'exécution abstraite* représentant les fonctions et le matériel activés par cette instruction.

a) Définition

Un graphe *d'exécution abstraite* est un graphe biparti ou :

- aux noeuds du premier type sont associés les éléments de mémorisation utilisés par l'instruction, aux noeuds du deuxième type sont associés les micro-opérations (ou fonctions) activées par l'instruction ;
- il y a un arc entre une microopération et un élément de mémorisation EM_i, si la microopération envoie le résultat du traitement dans EM_i, il y a un arc entre un élément de mémorisation EM_j et une microopération si la microopération traite la valeur contenue dans EM_j .
- Un graphe d'exécution abstraite a, en outre, la propriété suivante : les feuilles du graphe (sources et puits) sont toutes des noeuds du premier type (éléments de mémorisation).

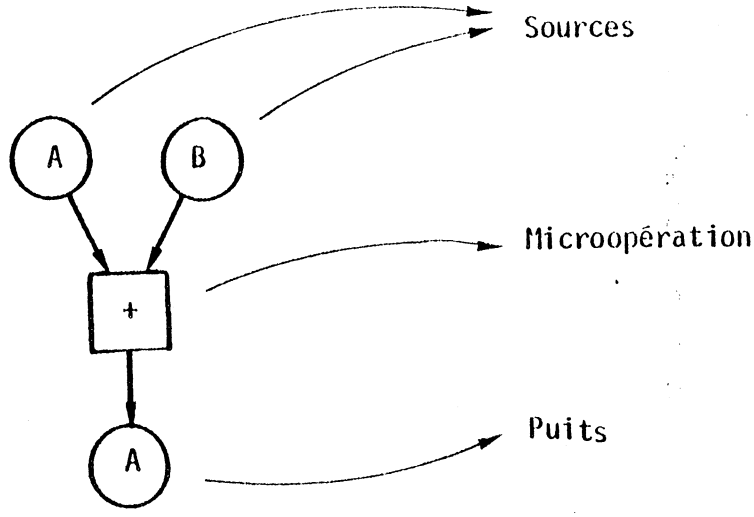
Un graphe d'exécution abstraite est dit *simple* s'il comporte une seule composante connexe ; sinon il est dit *multiple*.

Convention de représentation graphique : les noeuds du premier type sont représentés par des cercles, les noeuds du deuxième type par des carrés.

b) Exemples Instructions du microprocesseur Z80 .

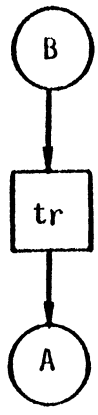
* Soit l'instruction ADD B qui réalise l'addition du contenu de l'accumulateur A et du registre B ; le résultat est stocké dans A .

Le graphe d'exécution abstraite associé est :



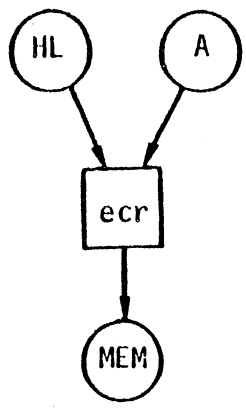
* Graphes d'exécution abstraite des instructions LD A,B LD (HL),A et ADDC A,B :

LD A,B



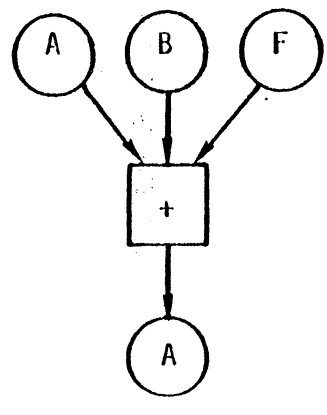
Charger le contenu du registre B dans l'accumulateur A .

LD (HL),A



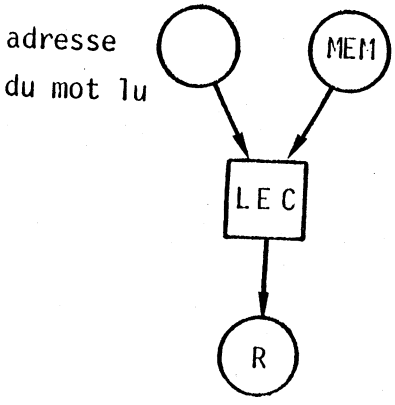
Charger en mémoire d'adresse HL le contenu de l'accumulateur A .

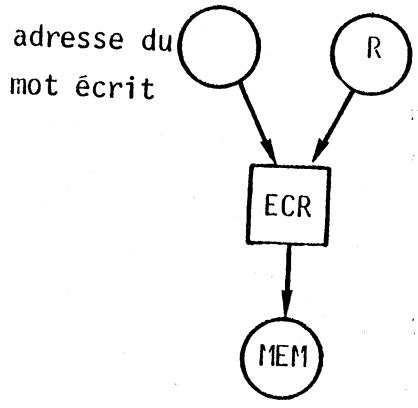
ADDC A,B



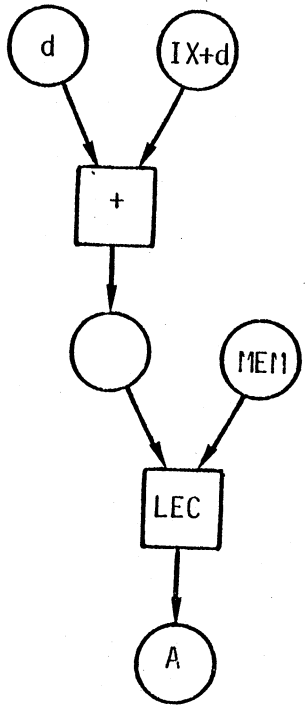
Addition entre le registre B et l'accumulateur A, avec retenue d'entrée et résultat dans l'accumulateur.

Convention de représentation : pour une microopération de Lecture ou Ecriture en mémoire, on convient que la branche de gauche (incidente au noeud Lecture ou Ecriture) provient de la variable "adresse du mot lu ou écrit".





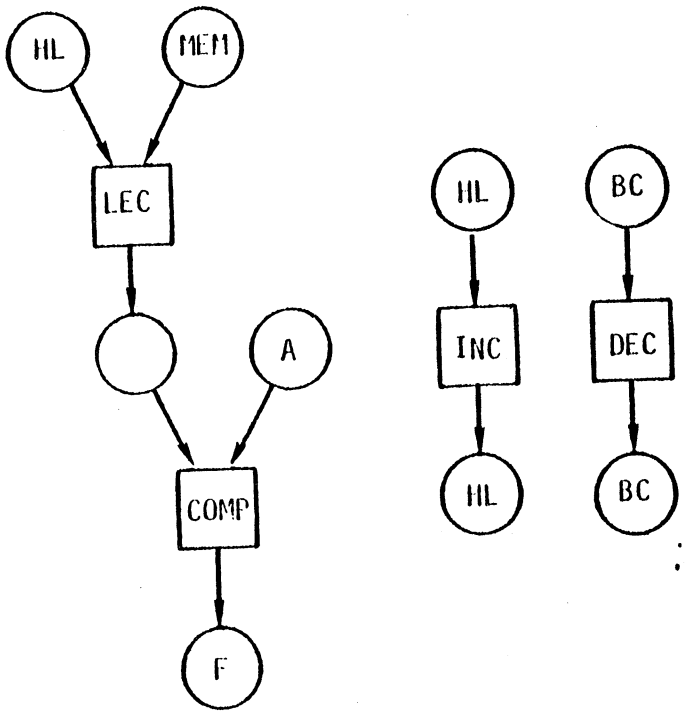
Graphe d'exécution abstraite à plusieurs couches



Instruction LD A,(IX+d)
Chargement du mot mémoire d'adresse (IX+d) dans l'accumulateur A .

Remarque : dans un graphe d'exécution abstraite un noeud de mémorisation intermédiaire peut ne pas être spécifié (non connu par l'utilisateur) ; il est alors représenté par un noeud vide (non étiqueté).

Graphe d'exécution abstraite à plusieurs composantes



Instruction CPI comparaison entre le contenu du mot mémoire d'adresse HL et l'accumulateur A; le résultat est chargé dans le registre d'indicateurs F .

B - ANALYSE DES INSTRUCTIONS

Dans cette étude préliminaire, on a recherché une classification des instructions selon leur complexité.

Chaque instruction étant représentée par son graphe d'exécution abstraite, on recherche des critères :

- de complexité, mesurée par la quantité de matériel activé par l'instruction,
- d'accessibilité, mesurée par la facilité d'accès de l'information de test pour cette instruction.

On utilisera deux niveaux d'analyse mettant en jeu des renseignements de plus en plus précis :

- sur les graphes non renseignés,
- puis sur les graphes renseignés.

B -1 Analyse structurelle des graphes d'exécution abstraite

Il s'agit de l'analyse de la "complexité" des graphes ne portant aucun renseignement sur la signification des noeuds. Il s'agit donc de considérer les graphes "nus".

On peut essentiellement définir :

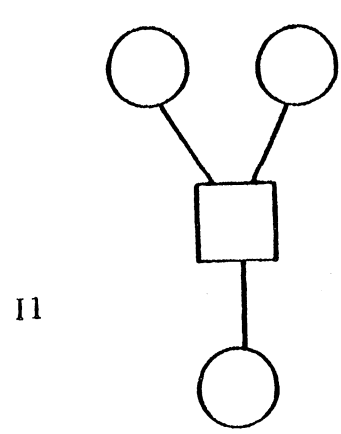
- une relation de dominance structurelle,
- des paramètres d'affinement structurels.

a) Relation de dominance structurelle

Définition

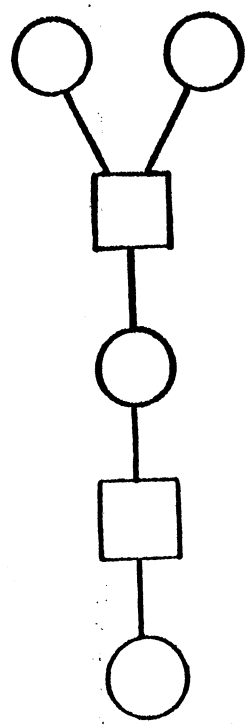
Le graphe d'une instruction I1 est structurellement dominé par le graphe d'une instruction I2 si le graphe de I1 peut être immergé dans le graphe de I2 (en théorie des graphes, on peut dire que le graphe de I1 est isomorphe à un sous-graphe partiel de I2).

Exemple :

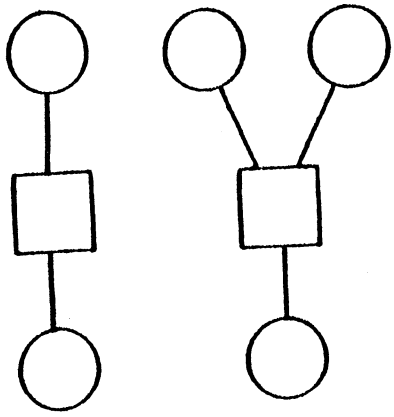


I1

est dominé par

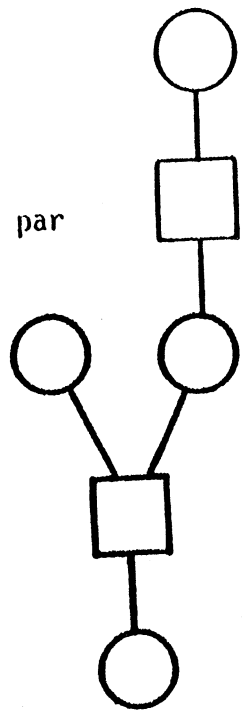


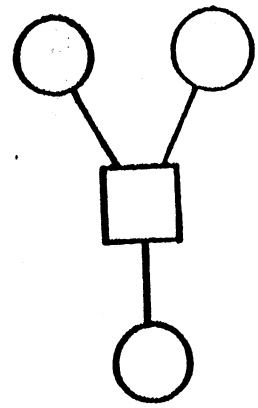
I2



I1

est dominé par





I2

b) Paramètres d'affinement

Ce sont les paramètres classiques de la théorie des graphes [Ber 63] ; on peut associer au graphe d'une instruction :

- le nombre d'arcs,
- le nombre de noeuds ou, plus précisément, le nombre de noeuds de l'un des types ou de chaque type,
- la profondeur ou nombre de couches de l'arborescence la plus profonde,
- le degré de multiplicité qui est le nombre de composantes connexes (graphe multiple).

B -2 *Analyse fonctionnelle des graphes*

Cette analyse se fait par rapport aux graphes renseignés et permet d'étudier l'accessibilité des instructions.

On définit comme précédemment :

- une relation de dominance fonctionnelle,
- des paramètres fonctionnels attachés aux noeuds : éléments de mémorisation et micro-opérations.

a) Relation de dominance fonctionnelle

Définition

- La *commandabilité* d'un élément de mémorisation traduit la facilité d'amener une information de test vers cet élément,
- L'*observabilité* d'un élément de mémorisation traduit la facilité d'observer un résultat de test issu de cet élément.

Les éléments de mémorisation peuvent être classés selon leur commandabilité et leur observabilité : on leur associe une valeur de commandabilité t et une valeur d'observabilité t'.

Un élément de mémorisation est dit de commandabilité i ($i > 1$) si l'information amenée à cet élément depuis l'extérieur doit passer par au moins un élément de commandabilité i - 1.

Un élément de mémorisation est dit d'observabilité j ($j > 1$) si l'information issue de cet élément doit transiter par au moins un élément d'observabilité j - 1 avant d'être observé à l'extérieur.

Le monde extérieur est dit de commandabilité 1 et d'observabilité 1 : il est générateur de l'information de test et observateur du résultat de test.

Exemple :

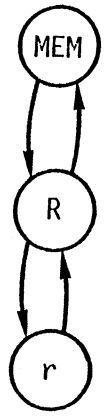
- La mémoire de travail et les périphériques, qui sont le monde extérieur sont directement commandables et observables.

$t = t' = 1$

- Les registres internes, pouvant communiquer directement avec l'extérieur, sont de commandabilité 2 et d'observabilité 2.

$t = t' = 2$

- Les registres internes ne pouvant communiquer avec l'extérieur que par l'intermédiaire d'un registre du type précédent sont de commandabilité 3 et d'observabilité 3.



$t = t' = 3$

- Un cas particulier est celui des valeurs immédiates qui, au moment de l'exécution d'une instruction, sont des valeurs internes au microprocesseur et sont définies comme étant de commandabilité $t = 0$. Le propre d'une valeur immédiate étant d'être une information d'entrée, il n'y a pas à définir de valeur d'observabilité.

- *L'accessibilité* d'une instruction est définie par la commandabilité de ses sources et l'observabilité de ses puits ; elle est représentée par un couple (t, t') où

  . t est la plus grande des valeurs de commandabilité des différentes sources,

  . t' est la plus grande des valeurs d'observabilité des puits.

l'instruction est dite (t, t')-accessible.

Remarque : cette première approche a donné lieu ensuite à l'étude plus générale de l'accès aux éléments de mémorisation (§ III section I).

Relation de dominance fonctionnelle

Une instruction (t_1,t'_1)-accessible est fonctionnellement dominée par une instruction (t_2,t'_2)-accessible si et seulement si $(t_1,t'_1) < (t_2,t'_2)$, sachant que

$$(a,b) < (c,d) \iff (a \leq c \text{ et } b \leq d) \text{ et } (a,b) \neq (c,d)$$

b) Paramètres fonctionnels

Des paramètres d'affinement sont associés aux feuilles du graphe et aux microopérations.

- Pour les feuilles : format de l'élément de mémorisation associé à la feuille (8 bits, 16 bits, ...) par rapport au format élémentaire e défini comme la taille de la plus petite information adressable en mémoire.
- Pour les microopérations : on peut essentiellement distinguer :
  . les microopérations effectuant un traitement ou manipulation de données,
  . les microopérations de transfert/lecture/écriture,

  et attacher un poids distinct à ces opérations.

Remarquons que le nombre d'opérandes nécessaires à la microopération est déjà pris en compte.

C - ORDONNANCEMENT DES INSTRUCTIONS POUR LE TEST

On recherche un ordre partiel ou total donnant un ordonnancement de test des instructions ; cet ordre est déduit des relations de dominance et des paramètres, structurels et fonctionnels.

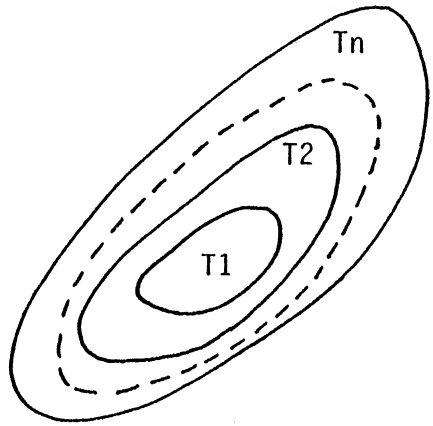
Deux organisations classiques peuvent être étudiées [Den 68] :

- l'approche structurelle progressive ou "start-small",
- l'approche descendante ou d'activation rapide.

C -1 L'approche start-small

Cette approche consiste à tester au préalable une petite partie de matériel, en général la plus petite quantité permettant de dérouler les tests ultérieurs (hardcore). Pour chaque test supplémentaire, on considère une partie du matériel ne dépendant que des parties précédemment testées (accès et observation de l'information de test).

Lorsqu'un test détecte une erreur, on en déduit que l'élément défectueux appartient au matériel rajouté pour ce test.



L'utilisation de ce type d'approche nécessite donc un ordonnancement des instructions face au test. Les avantages d'une telle approche sont les suivants :

- pour une instruction donnée, on ne teste, parmi le matériel et les fonctions activées par cette instruction, que la partie non encore testée par des instructions classées et donc testées précédemment (test progressif et adaptatif) ;
- facilité de génération et d'observation de l'information de test pour le matériel testé (consistence, propagation, masquage) ;
- information sur la cause de l'erreur détectée et localisation à un ensemble d'instructions près ;
- prise en compte possible des pannes multiples ;

a) Partition structurelle

Etant donné un ensemble d'instructions, la relation de dominance structurelle induit une partition de ces instructions en classes Co,...,Ci,... telles que :une instruction appartient à Ci si et seulement si elle n'est dominée par aucune autre instruction $\in$ (Co $\cup$ ... $\cup$ Ci - 1).

Affinement structurel

Etant donné un ensemble d'instructions, les paramètres structurels

- degré de multiplicité,
- profondeur,

des graphes d'exécution abstraite associés aux instructions, induisent une partition de ces instructions en sous-classes $\mathcal{C}_{k,p}$ telles que : une instruction appartient à $\mathcal{C}_{k,p}$ si et seulement si elle est de degré de multiplicité k et de profondeur p.

b) Partition fonctionnelle

Etant donné un ensemble d'instructions, la relation de dominance fonctionnelle induit une partition de ces instructions en blocs $\mathcal{B}o,\ldots,\mathcal{B}j,\ldots$ tels que : une instruction appartient à $\mathcal{B}j$ si et seulement si elle n'est fonctionnellement dominée par aucune instruction $\in \{\mathcal{B}o \cup \ldots \cup \mathcal{B}j - 1\}$

Affinement fonctionnel

Etant donné un ensemble d'instructions, les paramètres fonctionnels

- taille des éléments de mémorisation,
- type de microopération,

des graphes d'exécution abstraite associés aux instructions, induisent une partition de ces instructions en sous-blocs $\mathcal{B}_{e,t}$

- où e représente le format maximum des sources et des puits,
- où t indique le type de microopération : t = 1 pour une microopération de traitement, t = 0 sinon,

tels que : une instruction appartient à $\mathcal{B}_{e,t}$ si le format maximum de ses sources ou puits est égal à e et la microopération réalisée est de type t.

c) Ordonnancement des instructions pour le test

Dans une approche de type start-small, on obtient donc l'algorithme suivant d'ordonnancement des instructions pour le test :

- Partition structurelle sur l'ensemble des instructions du microprocesseur donnant l'ensemble ordonné {Co,...,Cn}. Les instructions de la classe C_i sont testées avant les instructions de la classe C_{i+1}.
- Pour chaque classe Ci ainsi définie, affinement structurel Ci = $\{C_{k,p}\}$ Les sous-classes sont ordonnées de telle sorte que : $C_{k,p}$ est testée avant $C_{k',p'}$ si et seulement si ($k < k'$) ou ($k = k'$ et $p < p'$)
- A l'intérieur de chaque sous-classe ainsi définie, partition fonctionnelle donnant l'ensemble ordonné : {Bo,...,Bm}. Les instructions d'un bloc B_j sont testées avant les instructions du bloc B_{j+1}.
- Pour chaque bloc B_j ainsi défini, affinement fonctionnel $B_j = \{B_{e,t}\}$

Les sous-blocs sont ordonnés de telle sorte que $B_{e,t}$ est testé avant $B_{e',t'}$ si et seulement si ($e < e'$) ou ($e = e'$ et $t < t'$).

Exemple :

L'ordonnancement des instructions dans l'optique d'un "Start-small" est illustré au moyen de graphes d'exécution abstraite du microprocesseur Zilog Z80 dans la figure 21 et la table 4.

Figure 21 : Partition structurelle et affinement

(à l'intérieur de la classe C1)

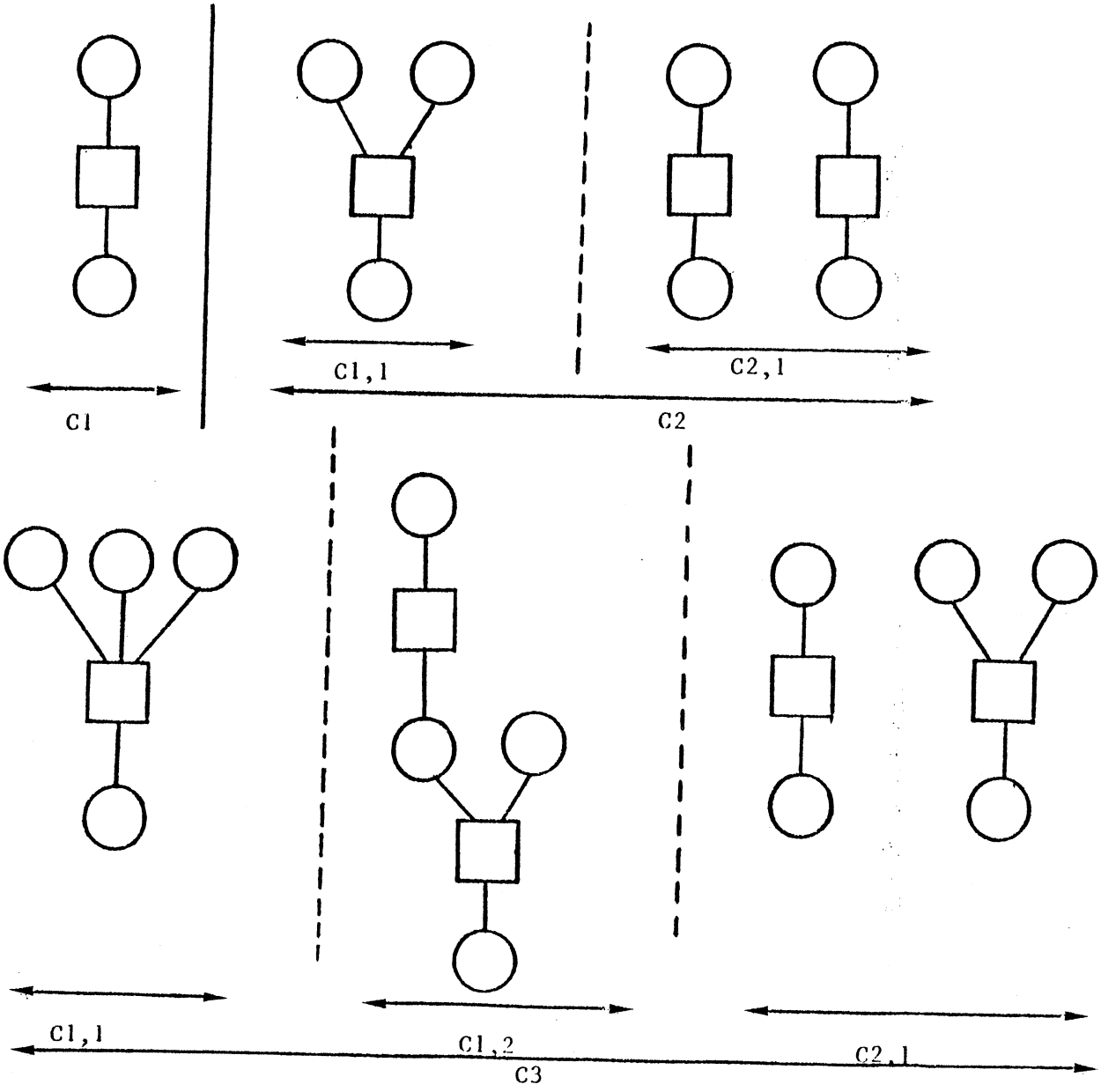


Table 4 : Partition fonctionnelle et affinement

(à l'intérieur de la classe C_1')

t, t'				

(0,2)
B_1
	LDr,n	$B_{1,0}$	LD dd,nn
			LD Ix,nn
			LD IY,nn
			JP nn
				$B_{2,0}$

(2,2)
B_2
	LDr,r'	$B_{1,0}$	LD SP,HL
			LD SP,IX
			LD SP,IY
			JP(HL)
			JP (IX)
			JP (IY)
				$B_{2,0}$

	INC r DEC r DAA CPL NEG CCF SCF RLCA RLC r RRC r SLA r SRA r SRL r	$B_{1,1}$	INC S	$B_{2,1}$
(2,3) B_3	LD R,A LD I,A			
(3,2) B_4	LD A,R LD A,I			

C -2 *Approche descendante (activation rapide)*

Une approche de ce type permet une vérification rapide du matériel, et est adaptée pour un système opérationnel dans lequel par exemple le microprocesseur a des périodes d'oisiveté.

L'objectif est alors de choisir un ensemble minimal d'instructions couvrant au moins une fois chaque élément de mémorisation et chaque micro-opération. Ce choix se fait en deux étapes :

a) Choix de l'ensemble dominant

Sur l'ensemble E des instructions du microprocesseur, on définit l'ensemble dominant $\mathcal{D}$ comme le sous-ensemble des instructions qui ne sont structurellement dominées par aucune autre instruction.

b) Choix de l'ensemble complémentaire

Lorsque l'ensemble dominant n'active pas au moins une fois chaque micro-opération et chaque élément de mémorisation, on complète cet ensemble par un sous-ensemble d'instructions I de $(E-\mathcal{D})$ de façon à assurer une couverture complète.

L'ensemble de test pour une telle stratégie, est alors constitué par $\mathcal{D} \cup I$.

Sur cet ensemble de test, l'ordre d'exécution des instructions est alors déterminé comme suit : une instruction $I1$ est exécutée avant une instruction $I2$ si et seulement si $(t_2,t_2') < (t_1,t_1')$.

Lorsque les mesures d'accessibilité (t,t') ne sont pas comparables, l'ordre est indifférent.

Exemple :

Considérons l'exemple du microprocesseur Zilog Z80 dont les graphes d'exécution abstraite peuvent être trouvés en [Vel 80b]

L'ensemble dominant $\mathcal{D}$ est constitué des sous-classes {6a, 6b, 6c, 6d, 6e} parmi les 22 sous-classes définies.

Les 19 instructions de cet ensemble constituent donc l'ensemble minimal de test dans l'optique d'une telle stratégie. Cet ensemble dominant ne couvrant pas tous les éléments de mémorisation et microopérations, il peut être complété par un ensemble de couverture C constitué des sous-classes {1, 2b, 4a}.

L'ensemble de test pour une telle stratégie comporte dons 75 instructions La table 5 montre le résultat de cette approche pour le Z 80.

	S-classe	Registres couverts	Micro-opérations couvertes
D	6a	A, IX , IY	ADDC, SBC,
	6d		SET, RESET, RR, RL
	6c	HL, A	RLD, RRD
	6d	BC, DE	LD
	6e	F, PC, SP	
C	1	I,R	INC,DEC,DAA,CPL, NEG,RLC,RRC,SLA, SRA,SRL,
	2b	A', B',.. H', L'	
	4a		AND,OR,XOR,CP,BIT, ADD,SUB

Table 5 : Approche descendante

D - ORGANISATION ET OPERANDES DE TEST

Dans cette étude préliminaire, les test de conformité et de balayage n'ont pas été séparés. Le test consistera à

- tester les instructions dans un ordre de complexité croissante ou décroissante (suivant l'approche chosie)
- balayer les points mémoire et les microopérations, par des vecteurs de test préétablis, au cours de la première instruction testée les utilisant.

D -1 Test d'une instruction

Le test d'une instruction I est réalisé par des séquences élémentaires comportant :

- l'initialisation de l'état réduit du microprocesseur,
- l'exécution de l'instruction I,
- l'observation de l'état réduit.

Ces trois pas sont itérés pour que durant le test de l'instruction soient appliqués :

- les vecteurs de test de conformité de l'instruction I (cf. Section I §V)
- les vecteurs de test de balayage des éléments de mémorisation et microopération utilisés par I et non encore testés.

L'initialisation et l'observation de l'état réduit sont réalisées par les séquences d'instructions correspondantes déterminées en Section I §IV)

D -2 Structure générale du programme de test

La stratégie de test détermine l'ensemble d'instructions à tester ainsi que l'ordre dans lequel elles doivent être testées.

Soit $\{I_1, I_2, \ldots, I_p\}$ cet ensemble ordonné d'instructions, le programme de test est alors composé des séquences de test de ces instructions. Les séquences correspondant aux instructions dont les graphes d'exécution abstraite ont la même complexité, déterminent un module de test.

La table 6 montre la structure générale du programme de test.

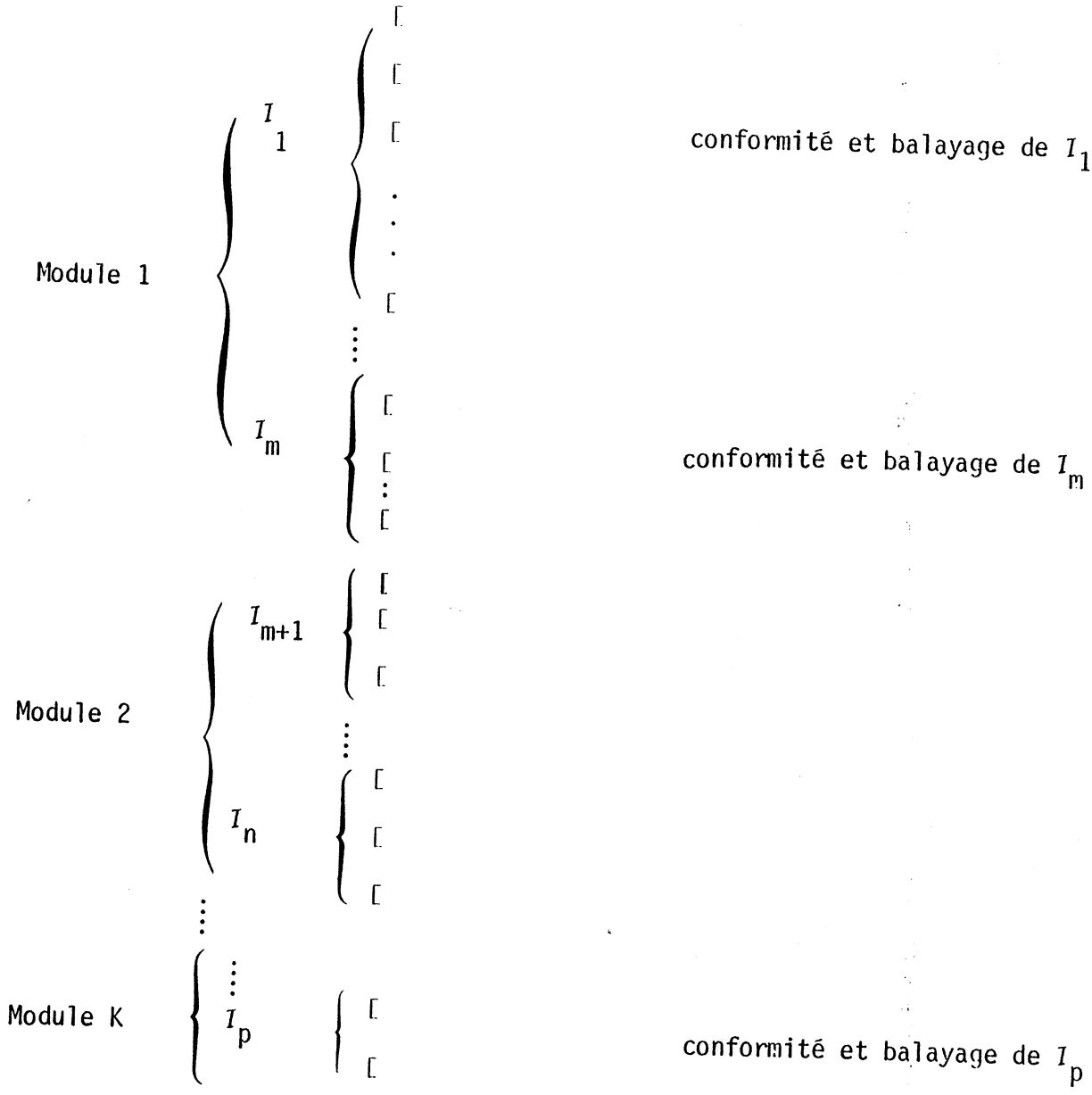


Table 6 : Structure générale du programme de test

Remarque : dans une optique "start-small" le programme de test comportera autant de modules que de classes de graphes d'exécution abstraite. Le nombre de séquences élémentaires sera évidemment plus important dans les premiers modules, dans la mesure où plus on avance dans le test moins il reste de EM et microopérations à tester. Pour les derniers modules ce nombre peut se réduire à une séquence élémentaire par instruction puisque seul le test de conformité sera à réaliser.

II - OUTILS LOGICIELS "ROBIN" [LIO 81]

Le logiciel ROBIN permet la génération des programmes de test en assembleur du microprocesseur testé, à partir de l'écriture dans un langage, appelé langage ROBIN, de séquences élémentaires paramétrées.

a) Le langage ROBIN

Le langage ROBIN permet

- l'écriture de primitives d'itération,
- la spécification des ensembles sur lesquels portent les itérations,
- l'écriture d'instructions assembleur paramétrées,
- l'utilisation de macro-instructions.

Exemple :

Test de l'instruction de chargement d'accumulateur avec valeur immédiate.

```
REG : (A,B)                                        Déclaration des ensembles
Valtest : (80,40,20,10,...,1)

iter : source (valtest)  puits (REG)               Primitive d'itération
     int : CALL INIT
     EXEC : LDA <puits>, #<source>                 Séquence élémentaire
     obsv : COPY OBSV                              paramétrée
f_iter
```

b) Le logiciel ROBIN

Le logiciel ROBIN génèrera la suite de séquences élémentaires en remplaçant les parties variables v par leurs valeurs courantes au fur et à mesure des itérations, gérées comme des boucles imbriquées. La séquence précédente en ROBIN donne, après expansion :

```
CALL INIT
LDAA   #80
OBSV

CALL INIT
LDAA   #40
OBSV
  ⋮
CALL INIT
LDAB    #1
OBSV
```

Exemple d'un programme complet écrit en ROBIN

```
titre "test du 6809"

declare:
%
          declarations d ensembles de registres
%
regab:(a,b)
regxs:(x,s)
%
          valeurs immediates
%
valim1:(1h,2h,4h,8h,10h,20h,40h,80h)
valim2:(1,2,4,8,10h,20h,40h,80h,100h,200h,400h,800h,1000h,2000h,4000h,8000h)
procedure:
%
%
          file  obs1
          staa adrsto
          stab adrsto-1
          tpa
          staa adrsto-2
          stx adrsto-3
          sts adrsto-5
adrsto set adrsto-7
          endf
algorithm:
%
*****************************************************************************
       classe1     s-classe1
*****************************************************************************
%
       section init
adrsto set 0fffh
          lds #2h
init1     clra
          clrb
          ldx #0
          tap
          rts
;
;         ldaa #n       ldab #n
;
iter: puits(regab) source(valim1)
   init:          jsr init1;
   exec:          lda(puits   #(source);
   obsv:          copy obs1;
fiter
;
;     ldx #n   lds #n
;
iter:  puits(regxs) source(valim2)
   init:          jsr init1;
   exec:          ld(puits  #(source);
   obsv:          copy obs1;
                  lds #2;
fiter
fin
```

d) Tableaux récapitulatifs des résultats de l'utilisation de ROBIN

* Le programme de test du 6800 comporte 7 classes (ou modules) dont les caractéristiques sont données dans la table 7 .

Classes	Nombre d'instructions ROBIN	Nombre d'instructions assembleurs générées	Taille du programme de test en octets	Taille des résultats en octets	Temps d'exécution
1	12	2 740	6 040	1 820	5 480 T
2	17	2 366	5 277	1 377	4 730 T
3	3	176	302	84	352 T
4	10	935	1 928	494	1 870 T
5	3	358	689	161	716 T
6	5	436	887	203	872 T
7	2	119	192	44	238 T
TOTAL	52	7 130	15 315	4 183	14 260 T

(T étant la période de l'horloge du microprocesseur sous test)

Table 7 : MC 6800

* Le programme de test du Z80 comporte 6 classes. La table 8 montre les caractéristiques des modules correspondants.

Classes	Nombres d'instructions ROBIN	Nombre d'instruction générées	taille du programme de test (en octets)	taille des résultats en en octets	temps d'exécution
1	12	6 880	10 004	7 782	21 920 T
2	24	14 664	24 051	12 945	117 312 T
3	3	400	569	384	3 200 T
4	7	774	1 356	617	6 192 T
5	8	863	1 672	737	6 904 T
6	4	244	419	102	1 952 T
TOTAL	58	23 825	38 071	22 567	190 600 T

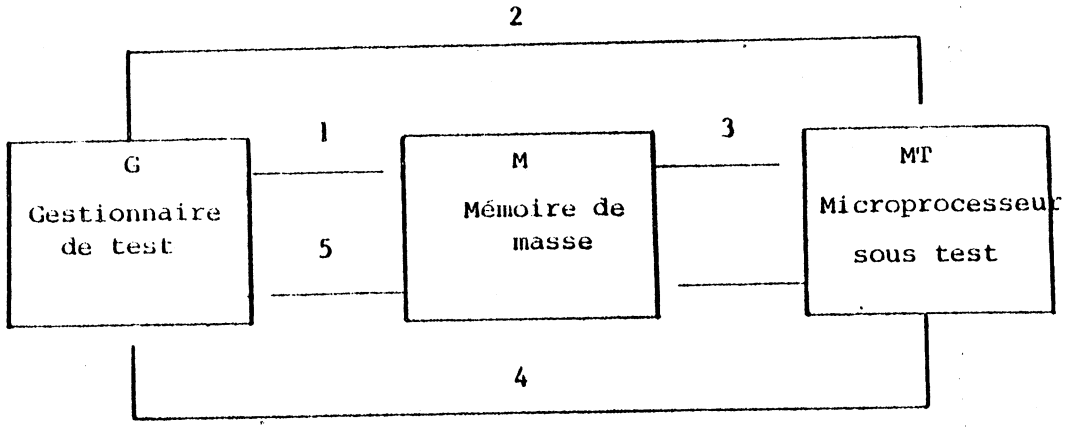
Table 8 : Z 80

III - CONCEPTION ET RÉALISATION D'UN TESTEUR

A - *OBJECTIFS*

D'après la méthode définie précédemment le système de test doit être capable de faire exécuter par le microprocesseur testé, le programme de test généré module par module(c.a.d. classe par classe), de récupérer les résultats obtenus et de les comparer aux résultats réputés bons.

L'architecture générale pour remplir ces fonctions est la suivante :



Le gestionnaire du test G

Il sert :

- au stockage des programmes de test et des résultats pré-établis
- à l'écriture en mémoire M des modules de test (1),
- à l'activation du microprocesseur sous test MT (2),
- à la récupération des résultats (5),
- à la comparaison des résultats obtenus à ceux pré-établis,
- à l'exploitation des résultats.

Le microprocesseur sous test MT

Il exécute les instructions stockées dans M sur un signal de G (3) et indique à G la fin d'exécution (4).

La mémoire M

Elle doit, d'après la méthode de test, couvrir tout l'espace d'adressage du microprocesseur testé (64K pour les microprocesseur 8 bits). L'accès à cette mémoire est partagé entre le gestionnaire de test et le microprocesseur sous test.

A -1 *Supports matériels utilisés*

Ce testeur a été réalisé d'une part dans un but de validation de la méthode de test et d'autre part pour permettre d'entreprendre des études ultérieures concernant le test et la fiabilité des composants.

Dans cette optique l'aspect performance de ce système n'a pas été un paramètre fondamental ; on s'est plutôt attaché à l'aspect évolutif et à la possibilité d'utilisation d'outils existants afin de minimiser le coût de mise en oeuvre. C'est pourquoi, les fonctions du système de test ont été réparties sur les supports matériels suivants :

- CII-HB 68
- Tektronix 8002
- Maquette conçue au Laboratoire de micro-informatique de Grenoble.

La figure 22 illustre les supports matériels utilisés.

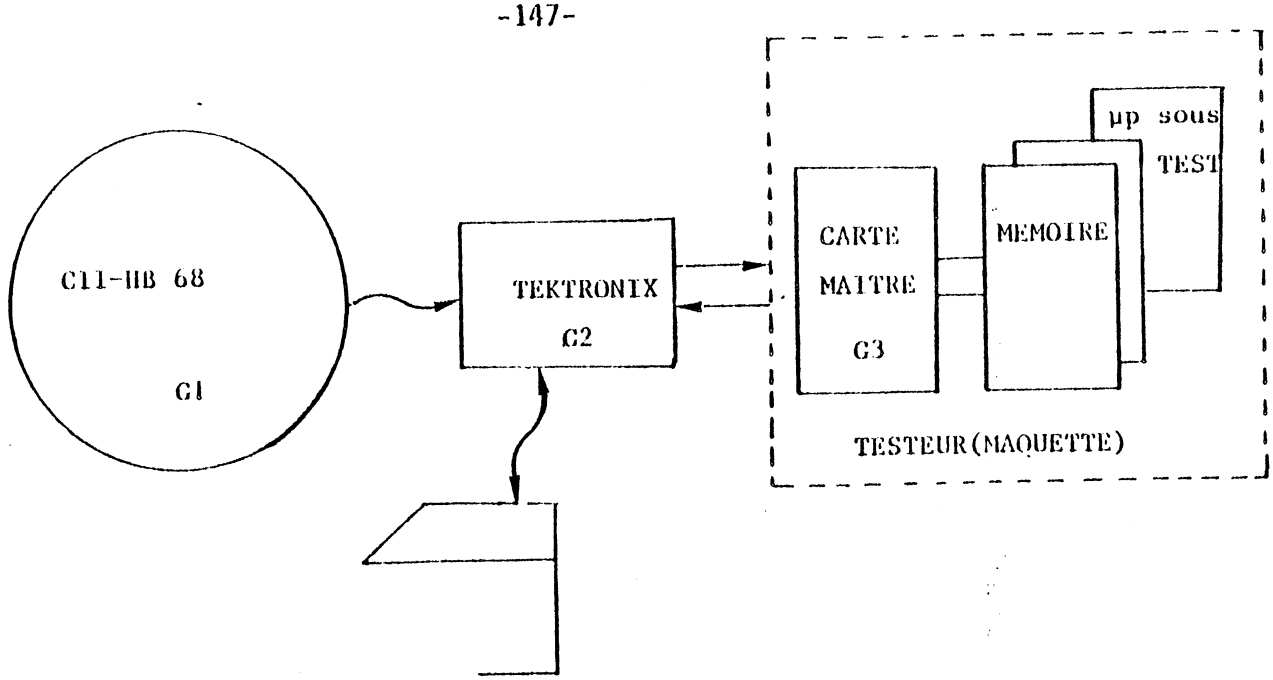


Figure 22 : Supports matériels utilisés.

Les fonctions G se trouvent ainsi éclatées en :

- G1 : Génération des programmes de test (exécution du programme "ROBIN" pour un microprocesseur donné et assemblage à l'aide des assembleurs croisés générés par le producteur GAGE (*)).
- G2 : Stockage sur disquettes des programmes objets obtenus en G1 et des résultats pré-établis.
  Dialogue avec l'opérateur.
  Gestion des liaisons avec HB 68 et la maquette.
  Exploitation des résultats.
- G3 : Gestion de l'exécution des programmes de test.

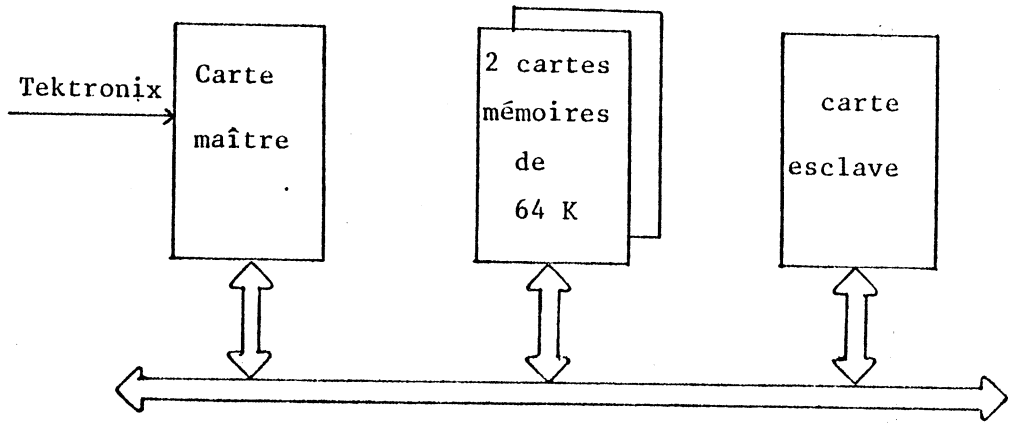
(*) GAGE : Générateur d'assembleur réalisé à l'atelier de Micro-informatique de Grenoble.

B - ARCHITECTURE DU TESTEUR

Le testeur se décompose en :

- une carte interface (ou carte maître)
- une carte microprocesseur sous test (ou carte esclave)
- deux cartes mémoires accessibles par la carte maître et par la carte esclave.

B -1 Schéma général du testeur



a) La carte microprocesseur sous test

Cette carte esclave contient :

- le microprocesseur à tester,
- des circuits de mise à niveau des signaux,
- des circuits d'adaptation aux bus de fond de panier,
- une horloge.

C'est une carte individualisée, spécifique à chaque type de microprocesseu

b) Les cartes mémoires

Elles contiennent au total 64K ce qui correspond à l'espace maximum d'adressage des microprocesseurs 8 bits existants (leur bus d'adresse étant de 16 fils).

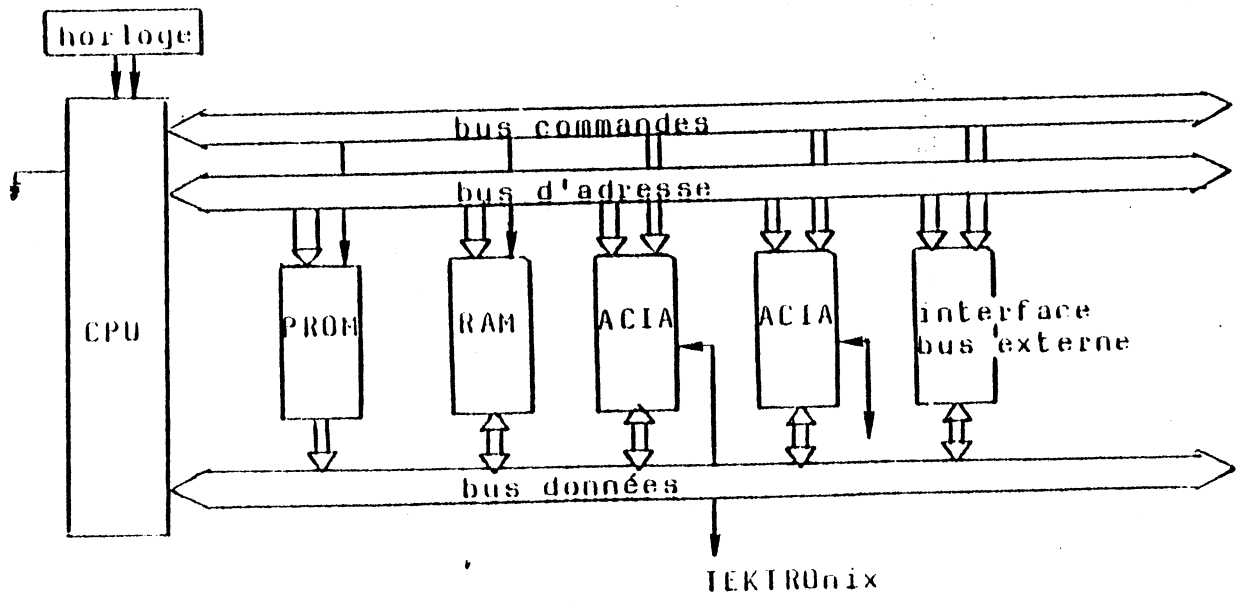
c) La carte interface

Cette carte assure la communication avec le Tektronix pour :

- le transfert des modules de test au Tektronix vers la mémoire du testeur,
- le transfert des résultats produits par le microprocesseur sous test vers la mémoire de masse du Tektronix.

Elle assure aussi la communication avec la carte sous test pour la gestion des signaux nécessaires au lancement et fin d'exécution d'un module de test.

B -2 Architecture de la carte mère



La carte comporte :

- un microprocesseur Z80
- deux circuits d'interface série (ACIA) permettant la communication avec le Tektronix et avec éventuellement d'autres périphériques,
- 4K octets de mémoire morte,
- 2K octets de RAM de travail,
- une interface avec le bus externe assurant la communication entre :
  . la carte maître et les cartes mémoires,
  . la carte maître et la carte esclave.

Les schémas détaillés sont présentés en Annexe 1.

L'adressage des 64K de mémoire externe ne peut évidemment pas se faire de manière directe. Il faut donc construire l'accès à cette mémoire.

On a résolu ce problème d'adressage de la manière suivante (fig. 23) :

- on adresse la mémoire externe comme un dispositif externe de stockage,
- l'interface aux bus externes sert à adresser la mémoire externe et comprend :
  . un verrou ("latch") pour les poids forts de l'adresse (ADRPF)
  . un verrou pour le poids faible de l'adresse (ADRPf)
  . un registre tampon ("buffer") pour les données (DATA)
  . un registre tampon de contrôle (CRTL)

Un accès à la mémoire externe se fera par :

- le chargement de l'adresse sur les latchs "adresse",
- la validation des signaux de contrôle et des données (en lecture ou écriture) par une instruction de lecture ou écriture dans le buffer DATA.

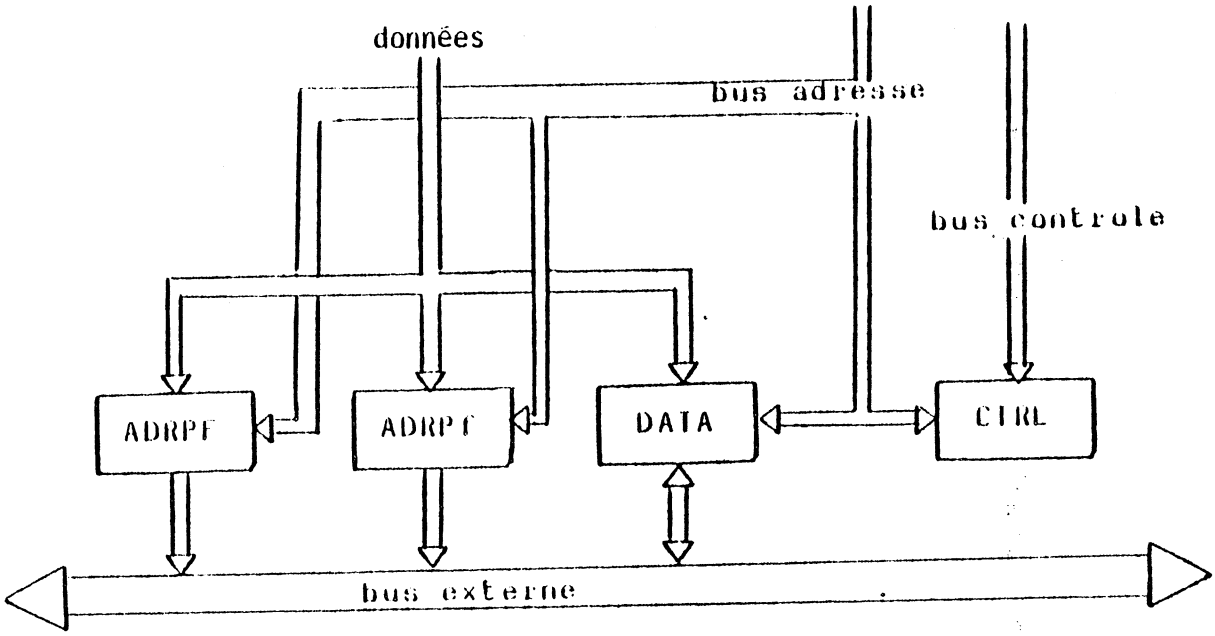


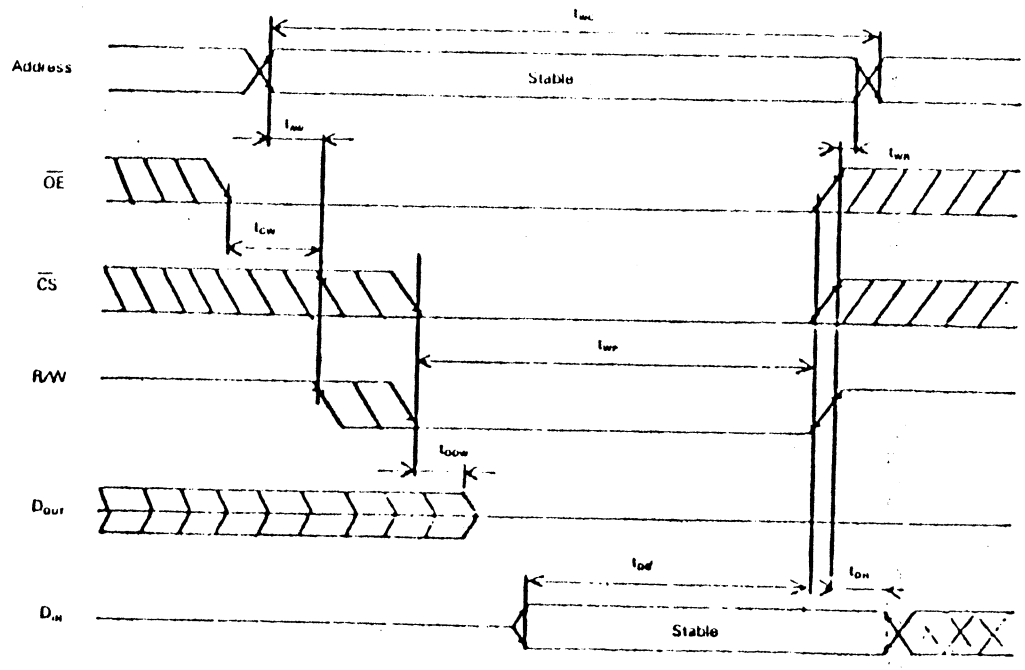
Figure 23 : Accès à la mémoire externe

L'accès à la mémoire se fera comme suit :

LD	A, PF	chargement dans ADRPF du poids fort
OUT	(ADRPF), A	de l'adresse
LD	A, Pf	chargement dans ADRPf du poids faible
OUT	(ADRPf), A	de l'adresse
LD	A, DONNEE	chargement de la donnée dans DATA et
LD	(DATA), A	validation du bus de fond de panier.

Cette manière de procéder permet de respecter le diagramme temporel d'accès aux mémoires employées. Ces diagrammes sont données dans la figure 24 .

Cycle d'écriture



Cycle de lecture

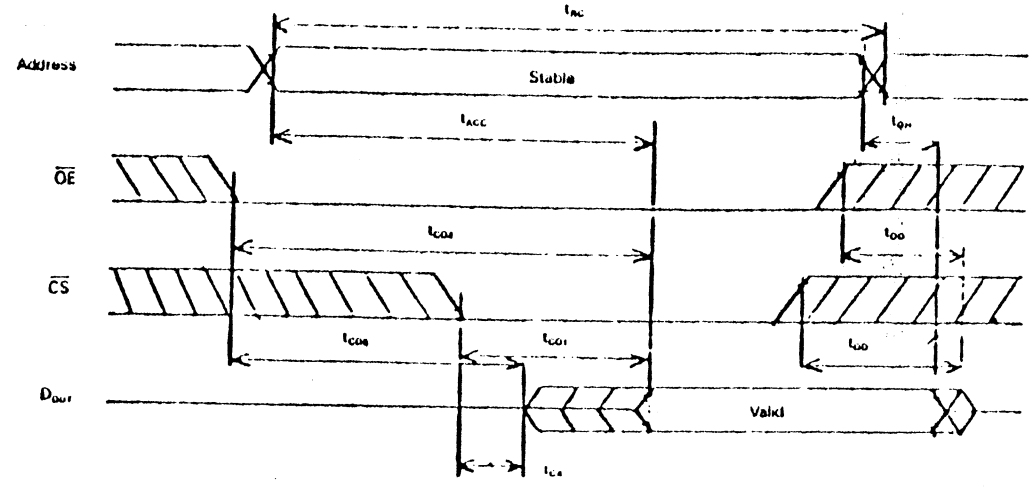


Figure 24 : Diagrammes temporels d'accès à la mémoire

B -3 *Bus de fond de panier* (Cf. Annexe 2).

Ce bus est divisé en trois parties :

- Signaux d'accès à la mémoire externe, partagés par les deux processeurs
  16 fils d'adresse
  8 fils données
  4 fils de contrôle
  c'est la carte maître qui impose la validation de ses propres buffers et ceux du microprocesseur sous test.
- Signaux de dialogue direct entre les cartes maître et sous test.
- Signaux propres à la carte maître (reset, transmission série, extensions ...).

C - LE MONITEUR [Dup 81]

C -1 *Fonctions du moniteur*

Le moniteur doit gérer le déroulement d'un programme de test. Ce déroulement s'effectue module par module (c.a.d. classe par classe).

Le test de chaque module comprend quatre phases (fig. 25)

- Chargement d'un module de test en mémoire du testeur.
- Lancement de l'exécution du module de test : le microprocesseur sous test exécute ce module de test et, en fin d'exécution, envoie un signal vers le microprocesseur maître.
- Sur reconnaissance du signal précédent, le microprocesseur maître récupère les résultats de test obtenus et les communique au Tektronix.
- La comparaison des résultats obtenus avec des résultats justes pré-établis se fera sur le Tektronix.

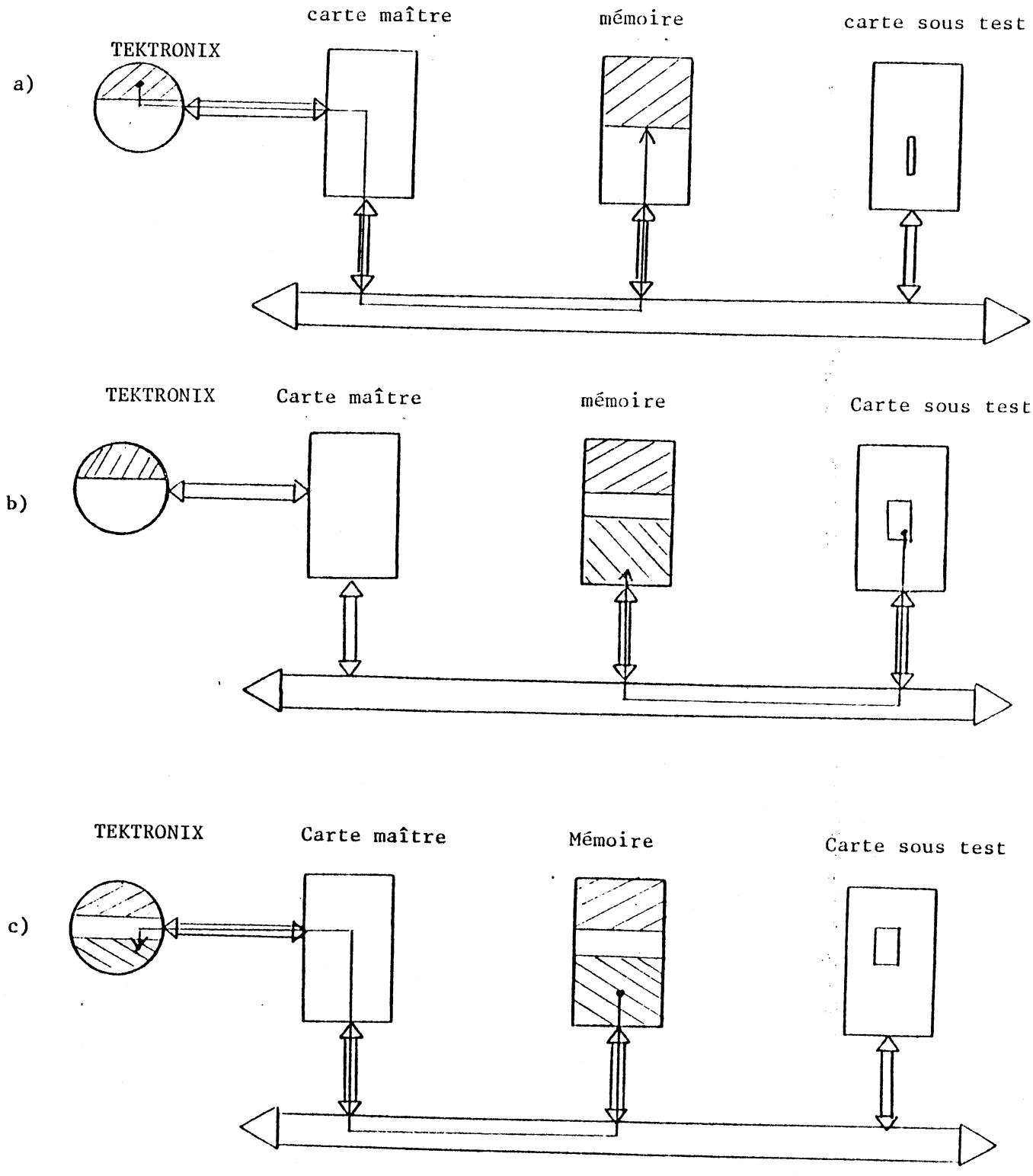


Figure 25 : Phases du test .

Les différentes tâches de gestion du test sont réparties entre le Tektronix et le testeur. La table 9 montre la répartition de ces tâches sur le Tektronix et la carte maître du testeur pendant le déroulement des diverses phases du test .

C -2 *Réalisation du moniteur*

L'approche logicielle a été faite de manière à assurer la plus grande souplesse d'adaptation.

Dans ce but, on a implanté en mémoire morte sur la carte maître un moniteur qui, en plus des fonctions propres à tout moniteur de microsystèmes (écriture/lecture de mémoire, points d'arrêts etc...), contient des commandes élémentaires supplémentaires capables d'assurer les fonctions nécessaires au test.

L'adaptation aux commandes de base reste à la charge d'un dispositif externe relié au testeur par une liaison série normalisée (V24).

Dans le cadre du développement actuel du testeur nous utilisons le Tektronix 8002 comme terminal intelligent.

a) *Définition des fonctions minimales du moniteur*

Ces fonctions minimales sont :

- l'accès à la mémoire externe du testeur
- le reset du microprocesseur sous test
- la détection du signal de fin d'exécution envoyé par le microprocesseur sous test
- la communication avec un dispositif externe.

Phase / Support	Ø0 Initialisation	Ø1 Communication de programme	Ø2 Exécution du module	Ø3 Communication des résultats	Ø4 Comparaison des résultats
TEKTRONIX	Activation du programme gérant le test Demande de RESET de la carte maître	- Opérateur : Choix du mode de fonctionnement - Transmission d'un module . lire fichier . transmettre au testeur un ordre "LOAD"	- Attendre signal de fin exécution envoyé par la carte maître	- Récupérer résultats en envoyant l'ordre "SAVE" - Stockage sur fichier	- Comparer les fichiers "résultats obtenus" et "résultats bons" Si égaux alors message OK et aller à Ø1; si erreur appel du programme de traitement d'erreurs - Procédure spéciale dans cas de dépassement du temps
TESTEUR	Utilisation des fonctions de base du moniteur + vérifications hardware (horloge,..) RESET	- Attendre ordre de TEKTRONIX - Si ordre "LOAD" alors ranger le programme en mémoire (à travers des buffers se trouvant sur la carte maître)	- Reset du microprocesseur sous test - Attendre le signal de fin envoyé par MT, contrôle de temps (chien de garde) - Avertir Tektronix de la fin d'exécution	- Lire mémoire de test aux adresses résultats sur ordre "SAVE" - Transmettre à TEKTRONIX	

Table 9 : Répartition des tâches du moniteur

Accès à la mémoire externe

Les commandes suivantes ont été réalisées :

- la commande (Z) permet d'initialiser l'ensemble de la mémoire externe du testeur avec une valeur de notre choix,
- la commande (M) permet de ranger des données, en mémoire externe du testeur à partir d'une adresse choisie,
- la commande (D) fournit l'image de la mémoire externe du testeur entre deux adresses.

Principe de Reset de microprocesseur sous test

Par convention le reset du microprocesseur sous test est le niveau 0 TTL. La commande (G) permet de générer le reset du microprocesseur sous test par programme.

Principe de la détection du signal de fin d'exécution du microprocesseur sous test

Comme pour le reset du microprocesseur, un signal est réservé pour indiquer la fin de l'exécution d'un programme.

Une instruction spéciale du microprocesseur testé (ex. HALT) génère un niveau sur une broche spécialisée du microprocesseur.

Ce signal est transmis à un buffer de la carte maître du testeur. Par convention le niveau 1 dans ce buffer indique que l'exécution est terminée.

Communication avec le dispositif externe

Cette communication doit permettre le transfert des programmes et résultats de test entre le testeur et l'extérieur.

Les commandes de transfert sont les suivantes :

- Une commande (LOAD) dont le rôle est de charger en mémoire externe du testeur des données arrivant sous enregistrement format MOTOROLA.
- Une commande (SAVE) effectuant la sauvegarde (c.a.d. la transmission sur la ligne série) de la mémoire, entre deux adresses, en enregistrement format MOTOROLA. Ce format est celui généré par les assembleurs "GAGE" utilisés sur le CII-HB 68.

Ces commandes de chargement et de sauvegarde ont été adaptées d'une part à l'utilisation d'un lecteur de ruban (démarrage du moteur du lecteur en début de programme et arrêt de celui-ci en fin de programme), et d'autre part à l'utilisation de Tektronix 8002.

b) *Etude des transferts d'information entre le Tektronix et le testeur*

Les programmes de test sont stockés en mémoire de masse du Tektronix. Le déroulement d'un programme de test nécessite donc les transferts suivants :

- chargement en mémoire externe du testeur d'un module de test résidant en mémoire du Tektronix,
- récupération des résultats de la mémoire du testeur par la mémoire de masse du Tektronix.

Nous avons prévu deux façons d'assurer ces transferts :

- En utilisant les primitives de communication de Tektronix (COMM).
- En gérant totalement cette communication à l'aide d'un programme utilisant, ce qui permet l'automatisation du déroulement des programmes.

La procédure de communication COMM au niveau transfert des données, est basée sur l'échange de blocs d'information entre deux interlocuteurs.

Au niveau fonctionnel la synchronisation du dialogue est faite comme suit :

- Après émission d'une ligne terminée par un retour chariot, le Tektronix attend un accusé de réception.
- En réception, le Tektronix reconnaît la fin d'une transmission de données par l'arrivée d'un indicateur. L'accusé de réception et l'indicateur de fin de transmission font partie des paramètres de la procédure COMM et sont réalisés par une séquence particulière de caractères. Nous avons choisi comme séquence de caractères : "OKAY1".

  Il est possible de dérouler un programme de test à l'aide des commandes précédemment définies avec intervention de l'opérateur à chaque phase du déroulement du test comme l'illustre l'exemple ci-après.

Exemple :

Déroulement d'un programme de test à l'aide des primitives système "COMM"

```
     UCOMM
COMM P=4F 4B 415931 T=Ø5          Activation procédure COMM

*K:28ØØ
*G:7FØ                            Lancement d'un programme utilisateur par le
                                  moniteur de base de la carte maître

#L: M1/1
*RIOT EDJ                         Commande "LOAD"

#G:                               Commande "GO"
#S:1ØØ 11F>RES1/1                 Commande "SAVE"
*RIOT*EOJ*

#*COMM*EOJ
UCOMM COMPLETED
```

C-3 *Automatisation du déroulement d'un programme de test*

a) Objectifs

Il faut :

- rendre transparente pour l'opérateur la succession des commandes de base décrites précédemment,
- assurer un dialogue de haut niveau avec l'opérateur permettant le choix de différents modes de fonctionnement,
- assurer la gestion des erreurs.

b) Moyens utilisés

Pour réaliser ceci, la procédure de communication a été remplacée par un programme utilisateur résidant sur le Tektronix et accédant directement aux ports d'entrées/sorties.

Ce programme utilisateur peut ainsi assurer l'enchaînement de l'envoi des commandes de base nécessaires ainsi que le traitement des réponses provenant du testeur, libérant ainsi l'opérateur pour un dialogue de plus haut niveau tel que le choix des modes de fonctionnement, du traitement des erreurs, ...

c) Les modes de fonctionnement

L'utilisateur a le choix entre trois modes de fonctionnement :

- *mode 1 ou mode de localisation* : l'utilisateur choisit un module à tester. Après l'exécution de celui-ci, l'opérateur retrouve le choix du mode de fonctionnement.
- *mode 2 ou mode automatique* : tous les modules appartenant à un programme de test sont exécutés en séquence sans intervention de l'opérateur.
- *mode 3 ou mode bouclage* : le programme de test est déroulé en entier, sans interruption et boucle sur lui-même.

d) Traitement des erreurs

Il y a erreur quand les résultats obtenus après test diffèrent des résultats pré-établis. La gestion des résultats revient à comparer un par un les octets constituant les fichiers "résultats pré-établis" et

"résultats obtenus". A chaque discordance un message d'erreur est inscrit sur la console de Tektronix.

Le contenu de ce message précise :

- le numéro de l'erreur
- le module de test où est située l'erreur
- l'adresse du résultat
- le résultat attendu et le résultat obtenu.

Ce message d'erreur est de la forme :

Numéro de l'erreur ERR"nnnn"	Nom du module Module "n"	Adresse du résultat ADR"nnnn"	Octet lu M="XX"	Octet attendu E="XX"

Simultanément les messages sont enregistrés sur un fichier d'erreurs permettant ainsi un traitement ultérieur.

Exemple de déroulement d'un programme de test

```
TEKDOS  Z80      VERSION 3.0

> TEST/1

> G
ATTENTE RESET SUR TESTEUR.


QUEL MODE DE FONCTIONNEMENT CHOISISSEZ VOUS?
POUR RENSEIGNEMENTS TAPER AIDE.


AIDE
MODE1=TEST SUR UN MODULE PARTICULIER.
MODE2=TEST SUR ENSEMBLE DES MODULES.
MODE3=BOUCLE SUR MODE 2.


QUEL MODE DE FONCTIONNEMENT CHOISISSEZ VOUS?
POUR RENSEIGNEMENTS TAPER AIDE.


MODE1
QUEL MODULE DE TEST CHOISISSEZ VOUS?


M1/1
ERR 00001 .MODULE 1   ADR 0102    M=66     E=00
ERR 00002 .MODULE 1   ADR 0103    M=66     E=00
ERR 00003 .MODULE 1   ADR 0104    M=66     E=00
ERR 00004 .MODULE 1   ADR 0105    M=66     E=00
ERR 00005 .MODULE 1   ADR 0106    M=66     E=00
ERR 00006 .MODULE 1   ADR 0107    M=66     E=00
ERR 00007 .MODULE 1   ADR 0108    M=66     E=00
ERR 00008 .MODULE 1   ADR 0109    M=66     E=00
ERR 00009 .MODULE 1   ADR 010A    M=66     E=00
ERR 00010 .MODULE 1   ADR 010B    M=66     E=00
ERR 00011 .MODULE 1   ADR 010C    M=66     E=00
ERR 00012 .MODULE 1   ADR 010D    M=66     E=00
ERR 00013 .MODULE 1   ADR 010E    M=66     E=00
ERR 00014 .MODULE 1   ADR 010F    M=66     E=00
ERR 00015 .MODULE 1   ADR 0110    M=66     E=00
ERR 00016 .MODULE 1   ADR 0111    M=66     E=00
ERR 00017 .MODULE 1   ADR 0112    M=66     E=00
ERR 00018 .MODULE 1   ADR 0113    M=66     E=00
ERR 00019 .MODULE 1   ADR 0114    M=66     E=00
ERR 00020 .MODULE 1   ADR 0115    M=66     E=00
ERR 00021 .MODULE 1   ADR 0116    M=66     E=00
ERR 00022 .MODULE 1   ADR 0117    M=66     E=00
ERR 00023 .MODULE 1   ADR 0118    M=66     E=00
ERR 00024 .MODULE 1   ADR 0119    M=66     E=00

FIN DE L EXECUTION DU MODULE 1 .00024  ERREUR.
```

IV - RÉSULTATS OBTENUS

Les programmes de test obtenus à partir de l'étude préliminaire ont été appliqués à 4 "lots" de microprocesseur MC 6800. Certains de ces microprocesseurs ont été rejetés comme "mauvais" :

- Soit par des utilisateurs pour qui le comportement du microprocesseur n'était pas celui spécifié par le manuel d'utilisation.
- Soit par l'application des tests fonctionnels et/ou paramètriques détectant des mauvais fonctionnements.

La méthode de test définie en Section II détermine, pour le microprocesseur MC 6800, des programmes de test comportant 9 modules :

- Un module initial, appelé hardcore (M_0)
- 7 modules de test (notés $M_1, M_2, \ldots, M_7$) correspondant aux 7 classes données en Annexe 4 .
- Un module de test des signaux d'interruption.

La procédure de test adoptée comporte deux phases :

- Une phase de *test* pendant laquelle les modules de test sont exécutés successivement et les erreurs éventuelles sont listées.
  Le microprocesseur testé est déclaré BON , aucune erreur n'est détectée sinon il est déclaré MAUVAIS.
- Une phase de *diagnostic*. Pour chaque erreur on cherche d'abord la séquence élémentaire (INIT, EXEC, OBSV) dont l'exécution a produit le mauvais résultat. Une fois cette séquence localisée les instructions "fautives" peuvent être listées. Cette localisation ne s'avère possible que dans certains cas :

- erreurs dans le module 0
- nombre d'erreurs réduit
- erreurs répétitives.

Elle peut donner lieu à un diagnostic fin obtenu par *affinements successifs* :

1) Pour chaque instruction fautive peut être établie une liste d'hypothèses sur l'origine de l'erreur. Ces hypothèses concernent en général des mauvais fonctionnement des unités telles que l'UAL les registres, l'unité de contrôle, ...

2) Pour une hypothèse donnée, une séquence élémentaire est déterminée. L'exécution de cette séquence par le microprocesseur sous test permet de confirmer ou rejeter l'hypothèse.

3) Dans le cas où l'hypothèse est rejetée le pas 2 est répété. (choix et vérification d'une nouvelle hypothèse).

4) Si l'hypothèse est confirmée, la cohérence est alors étudiée ; il s'agit de vérifier les autres instructions pour lesquelles l'erreur devrait se manifester.

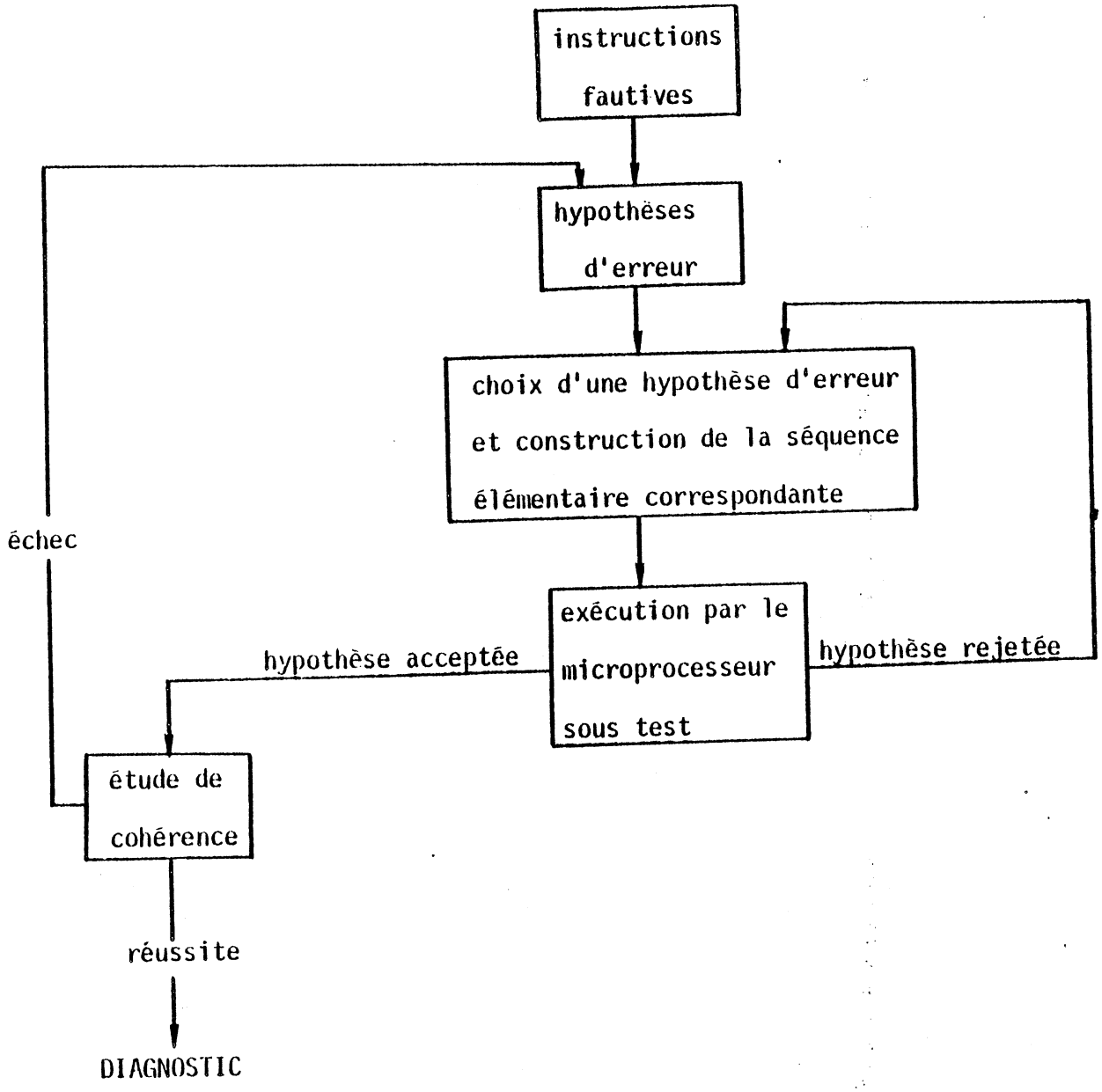


Figure 26 : Etapes d'obtention du diagnostic

Remarque : Le système de test a été muni d'un moniteur fournissant des facilités de "debugging" (inspection et modification des mots mémoire, insertion de points d'arrêt,...). Les séquences d'affinement peuvent alors être aisément mises en oeuvre.

Lot 1

Ces microprocesseurs ont été fournis par la Société EMD. Des tests fonctionnels et paramètriques ont été appliqués par le département de test de cette Société permettant la détection d'erreurs pour certains des microprocesseurs de ce lot.

La table 10 présente le diagnostic obtenu par application des programmes de test construits d'après la stratégie "start-small".

µP	Diagnostic	
1	BON	Aucune erreur détectée dans les conditions normales de fonctionnement.
2	MAUVAIS	N'accepte pas RESET
3	MAUVAIS	Localisation impossible
4	MAUVAIS	" "
5	MAUVAIS	Instructions de branchements conditionnels Ligne V collée à Zéro dans le circuit d'évaluation de la condition de branchement
6	MAUVAIS	Instruction ASR (décalage arithmétique à droite) Pas de conservation du bit de poids fait. A la place les circuits en panne charge le OU du bit de CARRY et de bit de Poids Forts
7	MAUVAIS	Localisation impossible

Table 10

Séquences d'affinement

Microprocesseur n°5

Les instructions fautives sont les branchements conditionnels relatifs BGE, BLT, BVS et BVC.

Le matériel mis en jeu est pour ces instructions le registre des indicateurs, le circuit d'évaluation des conditions, l'additionneur,... Par affinements successifs le diagnostic suivant a pu être établi : la ligne reliant l'indicateur V et le circuit d'évaluation des conditions est collée à 0 (Figure 27).

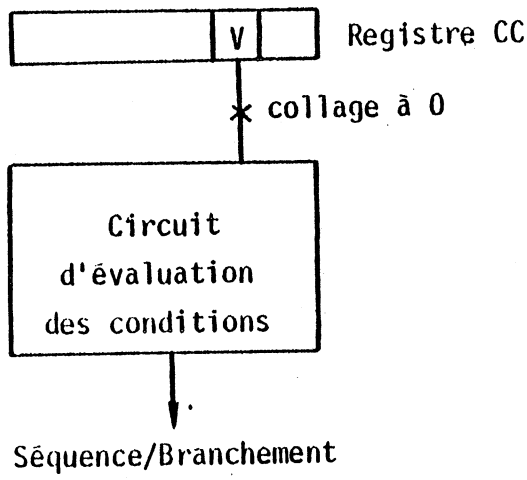


Figure 27

Cohérence : Les branchements conditionnels BGT et BLE utilisent aussi des conditions où V intervient. Ces instructions ont aussi été vérifiées, l'hypothèse étant confirmée.

La figure 28 représente les différentes hypothèses qui ont été vérifiées pour aboutir au diagnostic.

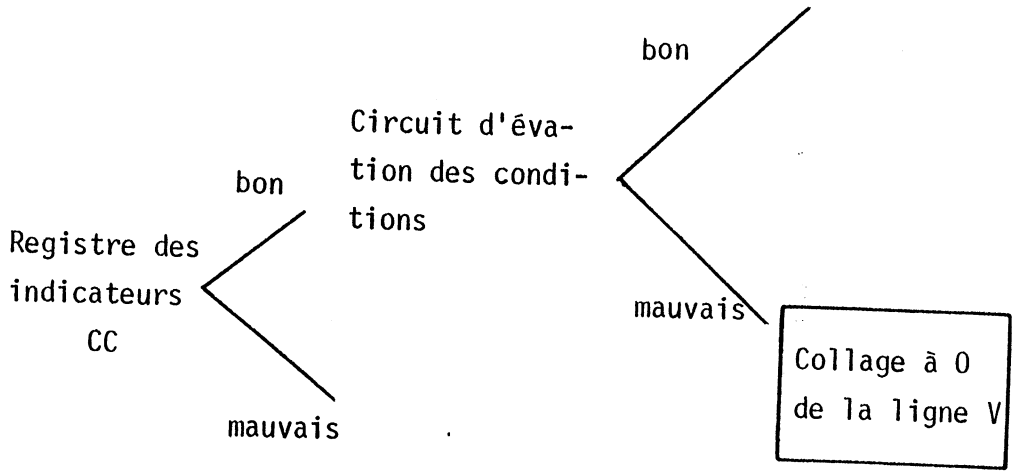
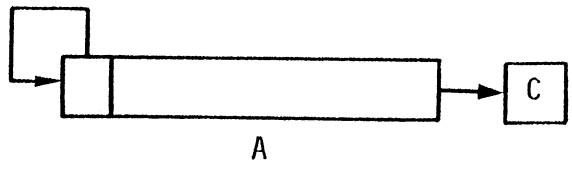


Figure 28 : Diagnostic et affinements pour le µP 5

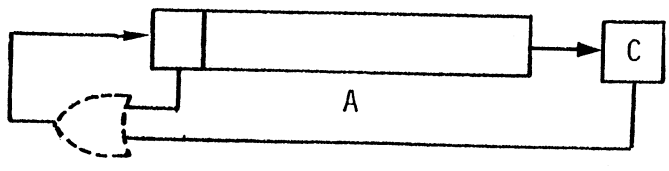
Microprocesseur n°6

L'instruction fautive est ici ASRA décalage arithmétique à droite. Le matériel mis en jeu est l'opérateur de décalage, l'accumulateur A, le registre d'indicateurs (bit) ....

Par affinements successifs l'hypothèse *opérateur de décalage mauvais* a été vérifiée. Plus précisément, lors de l'instruction ASRA, le bit de poids fort au lieu d'être conservé est chargé avec le OU entre ce bit et le bit de retenue C



Instruction ASRA



Microprocesseur 6

Lot 2

Ces microprocesseurs ont été envoyés à l'IMAG par des utilisateurs.

µP	Diagnostic	
1	MAUVAIS	Localisation impossible
2	BON	
3	MAUVAIS	Instructions de transfert avec A
4	MAUVAIS	Instructions PUSHB et branchements conditionnels

Table 11

Séquences d'affinement

Microprocesseur n°3

Des erreurs apparaissent dans le module 0. Lorsque l'accumulateur A est accédé. On suspectera alors les instructions de chargement et stockage de A ainsi que l'accumulateur A lui-même.

L'exécution des séquences d'affinement correspondantes (Figure 28) permet d'établir le diagnostic:

Lors de l'exécution des instructions de chargement de l'accumulateur A (LDAA n, LDAA m, LDAAn,X, TAP) la donnée FO est systématiquement chargée dans A.

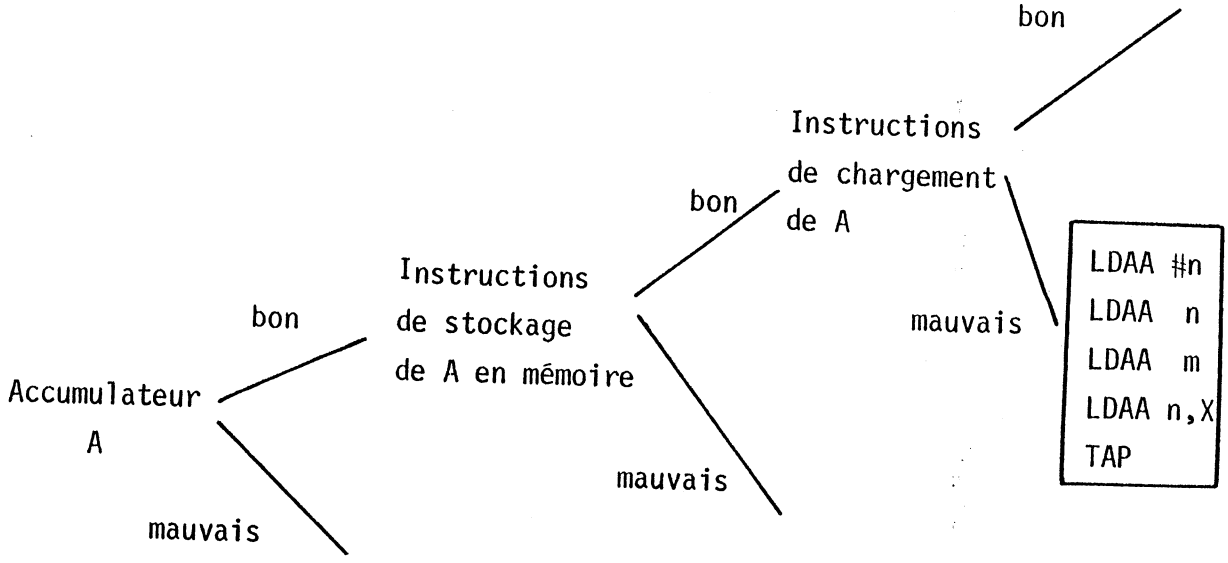


Figure 29 : Diagnostic et affinements pour le µP 3 du lot 2 .

Microprocesseur n°4

Instructions fautives PUSHB mise en pile externe de l'accumulateur B et branchement conditionnels.

Des séquences d'affinements correspondantes permettent d'établir le diagnostic :

-*Lors de l'exécution de l'instruction PUSHB la valeur FF est stockée en mémoire indépendamment du contenu de B; collage à 1 de la ligne V du circuit d'évaluation de condition.*

Lot 3

Ces microprocesseurs ont été installés dans un prototype d'émulateur (MAD), développé à l'IMAG, lors de sa mise au point. Cinq microprocesseurs ont ainsi subi des conditions électriques extrêmes, ce qui a donné lieu à des fonctionnements anormaux.

Pour ces cinq microprocesseurs, les programmes de test détectent des erreurs. La localisation de la faute est impossible, mais une analyse détaillée du nombre et type d'erreurs obtenues, ont permis de trouver par affinement que : ces microprocesseurs provoquent une écriture intempestive en mémoire lors de l'exécution des instructions qui utilisent la pile (JSR, WAI, RET).

Lot 4

Ce quatrième lot est composé de microprocesseurs sortis de la chaîne de fabrication qui devaient subir un test de réception d'entrée chez EMD. Aucun test (mis à part des tests de fin de fabrication) n' a été appliqué auparavant.

Les résultats obtenus sont montrés dans la table 12.

µP	Diagnostic
1	BON
2	MAUVAIS Pas de RESET
3	BON
4	BON
5	BON
6	MAUVAIS Modification intempestive de mots mémoire
7	BON
8	MAUVAIS Pas de RESET
9	MAUVAIS Pas de RESET
10	BON
11	MAUVAIS Pas de RESET
12	MAUVAIS Modification intempestive de mots mémoire
13	BON
14	BON
15	BON
16	BON
17	MAUVAIS Modification intempestive de mots mémoire

Table 12

Les diagnostics donnés pour les microprocesseurs des lots 1 et 4 ont été comparés à ceux obtenus par le département de test de EMD, par application d'une séquence de test fournie par MOTOROLA, mise en oeuvre sur un testeur TEKTRONIX 3270.

Bien qu'une vingtaine de circuits ne soit pas un nombre suffisant statistiquement pour évaluer l'efficacite de la méthode, la comparaison des deux diagnostics a été couronnée de succès et a encouragé la poursuite de ce travail. En effet :

- tout µP trouvé BON sur le testeur IMAG, l'est aussi sur le 3270,
- pour certains µP trouvés BONS sur le 3270 des erreurs (écriture intempestive en mémoire) ont été détectées ,
- en ce qui concerne les autres µP (trouvés MAUVAIS sur les deux testeurs) la séquence Motorola n'est pas prévue pour la localisation fine. Les diagnostics donnés après affinements, peuvent constituer une aide importante aux recherches sur la cause physique des erreurs et à l'amélioration des séquences de test.

BILAN ET PERSPECTIVES

Une étude préliminaire relativement complète a concerné la méthode de test des microprocesseurs ainsi que l'équipement de test associé (Section II). Elle a permis de prouver l'efficacité de cette approche et d'orienter les choix définitifs devant permettre d'atteindre l'objectif visé : la génération automatique des programmes de test de microprocesseurs. Le tableau récapitulatif ci-dessous montre les évolutions et les choix définitifs :

	Description	Méthode de test	Logiciel	Equipement de test
Etude préliminaire	Graphe d'exécution abstraite des instructions	.Classification par complexité .Balayage et conformité imbriqués	ROBIN : logiciel d'aide à l'écriture des programmes de test	Minimal
Version finale	Langage de description (transfert registre)	.Classification par type d'instruction(transfert, manipulation) .Modularité : balayage et conformité séparés .Test des signaux complémentaires	"GAPT" Générateur Automatique des Programmes de test	Hybride

On notera essentiellement

* Au niveau de la description, une évolution vers une description en langage formalisé permettant un passage aisé à la génération automatique.

* Au niveau de la méthode de test, une évolution vers une structure plus modulaire, fondée sur les types d'instruction et séparant soigneusement la conformité et le balayage.

* Au niveau de l'équipement de test, l'environnement minimal utilisé dans l'étude préliminaire a été satisfaisant au niveau des tests de conformité et de balayage mais il a été abandonné au profit de l'environnement hybride qui permet le test des signaux et le compactage des résultats.

En conclusion, cette étude a permis d'affirmer que le test comportemental, à condition d'être défini de façon claire et rigoureuse, est possible et efficace.

D'autre part, si ce test est formalisé à partir d'une description adéquate, la génération automatique de programmes de test est possible. Elle permettra de créer des bibliothèques de programmes de test des microprocesseurs du marché, et rendra sans doute des services importants aux usagers.

Enfin, le travail fait sur la mise en oeuvre du test, indique que les équipements de test doivent évoluer : un testeur classique (broche à broche) n'est pas adapté au test comportemental des circuits réalisant des fonctions complexes (microprocesseurs, périphériques,..).

Un test usager nécessite un testeur orienté "fonctionnement du circuit"; le testeur hybride définit dans ce travail nous semble correspondre à ces besoins.

BIBLIOGRAPHIE

[BEL 81] C. BELLON, J.M. GOBBI, G. SAUCIER,
"Hardware description levels and test for complex circuit"
Congrès DAC, Nashville, July 1981.

[BER 63] C. BERGE,
"Théorie des graphes et ses applications"
Edition DUNOD Collection Universitaire de Mathématiques,
Paris, 1963.

[CAS 77] McCASKILL R.,
"MPU Devices testing approachs. Tutorial on LSI testing"
Compcon spring 77, SAN FRANCISCO, USA, Feb. 1977.

[CHI 76] CHIANG A.C. and McCASKILL R.,
"Two new approachs simplify testing of microprocessors"
Electronics, Jan. 1976

[CLA 79] CLARY J.B., SACANE P.A.,
"Self test computers"
Revue Computer, Oct. 1979.

[DEL 80] DELAUNAY M., GRABOWIEKI J.F.,
"Utilisation du transformateur de grammaire LL (1)"
Rapport de Recherche 234, IMAG, Grenoble, Sept. 1980.

[DEN 68] DENT J.J.,
"Diagnosis engineering requirements"
Proc. AFIPS SJCC, 1968.

[DUP 81] DUPIN B.,
"Etude et réalisation du moniteur d'un système de test de microprocesseurs"
Rapport de stage, Institut de Programmation de Grenoble, Juin 1981

[EIC 78] EICHELBERGER E.B., WILLIAMS T.W.,
"A Logic Design Structure for LSI Testability"
Computer Science Press. Inc., 1978.

[FEE 77] FEE W.G.,
"Memory testing"
Compcon 77, Feb. March 1977.

[FOU 81] J.L FOURNIER , A. GEORGES , J.L RAINARD
"Analyse dynamique de circuits integrés VLSI, par stroboscopie au microscope à balayage électronique"
Rapport interne CNET, 1980

[FRA 81] FRANK E.H., SPROULL R.F.,
"*Testing and Debugging Custom Integrated Circuits*"
Dept. of Computer Science, C.M.U., Feb. 1981.

[FRO 77] FROHWERK R.A.,
"*Signature analysis : a new digital field service method*"
Hewlet-Packard Journal, May 1977.

[GOR 77] GORDON G., NADIG H.,
"*Hexadecimal signatures identify troublespots in microprocessor systems*"
Electronics, March 1977.

[HAY 80] HAYES J.P., McCLUSKY E.J.,
"*Testability considerations in microprocessors based design*"
Revue Computer, March 1980.

[HUS 75] HUSTON R.,
"*Microprocessor testing. A testing turnaround-smart DUT RUNS the tester*"
Fairchild System Technical Bulletin, Feb. 1975.

[KAI 79] KAI HWANG,
"*Computer arithmetic, Principles, architecture and design*"
John Wiley & sons inc. 1979.

[KIZ 80] KIZIS R.E., HERBERY R.T.,
"PLT Language and Languages Processing System"
IEEE Test Conference, 1980.

[KUN 65] KUNTZMANN S.,
"Algèbre de Boole"
Editions DUNOD, Paris, 1965.

[LIN 80] LIN M.G., WEITZ M., YUEN A.K., ROSE K.,
"Testing the 8086"
IEEE Test Conference, USA, Cherry hill, Nov. 1980.

[LIO 81] LIOTHIN A.,
"Conception d'un générateur de programmes de test de micro-processeurs"
Thèse CNAM, Grenoble, Feb. 1981.

[MIT 77] MITCHELL K.,
"Signatures analysis"
Electronics Industry, Nov. 1977.

[MUE 81] MUEHLDOIF E.I., SAVKAR A.D.,
"LSI Logic testing - An overview"
IEEE Transaction on Computers, Vol. 30, n°1, Jan. 1981.

[NGU 81] NGUYEN QUAND NA
"Programme de test fonctionnel de mémoires"
Rapport IMAG, Grenoble, Juin 1981.

[ROB 79] ROBACH Ch.
"Test et testabilité de systèmes informatiques"
Thèse d'état, Université de Grenoble, Juin 1979.

[ROB 80a] ROBACH Ch.,
"Test et fiabilité des microprocesseurs"
Colloque International sur la Fiabilité et la Maintenabilité,
Perros-Guirec/Trégastel, France, Sept. 1980.

[ROB 80b] ROBACH Ch., SAUCIER G.,
"Microprocessor functional testing"
IEEE Test Conference, USA, Cherry Hill, Nov. 1980.

[ROB 81] ROBACH Ch., SAUCIER G.,
"Le test des microprocesseurs et de systèmes à microprocesseurs état de l'art et perspectives"
L'onde Electrique, March 1981.

[ROT 68] ROTH J.P.,
"Diagnosis of automata failure : a calculus and a method"
IBM Journal R & D, Vol. 10, pp 278-291, 1968.

[SAU 81] SAUCIER G.,
"Conception sûre et choix de l'architecture d'un circuit à très haute intégration"
Rapport, IMAG, Dec. 1981.

[NIC 80] NICKEL V.,
"VLSI - The inadequacy of the Stuck at fault model"
IEEE Test Conference, USA, Cherry Hill, Nov. 1980.

[PAR 82] PARKER K.P., WILLIAMS T.W.,
"Design for testability. A survey"
IEEE Transaction on Computers, Vol C-31, n°1, Jan. 1982.

[PAS 80] PASTUREL M.,
"What makes Pascal a modern test language"
IEEE Test Conference, USA, Cherry Hill, Nov. 1980.

[PAT 80] PATRIE R.D.,
"The macro assembleur as a tool in LSI pattern generation"
IEEE Test, Conference, USA, Cherry Hill, Nov. 1980.

[POA 63] POAGE J.P.,
"The derivation of optimum test for logic circuits"
Ph. D Princeton University, 1963.

[ROB 78] ROBACH Ch., SAUCIER G.,
"Dynamic testing oh control units"
IEEE Transaction Computer, July 1978.

[SRI 81] SRIDHAR T., HAYES J.P.,
"A functional approach to testing bit-sliced microprocessors"
IEEE Transaction on Computers, Vol. C-30, n°8, Aug. 1981.

[THA 78] THATTE S.M., ABRAHAM J.A.,
"A methodologie for functional level testing of microprocessors"
Fault-tolerant computing Symposium, Toulouse, Juin 1978.

[THA 79] THATTE S.M., ABRAHAM J.A.,
"Test generation for general microprocessors architecture"
Fault-tolerant Computer Symposium, Juin 1979.

[THE 78] THEVENOD-FOSSE P., DAVID R.,
"A method to analysis ramdom testing of sequential circuits"
Digital Processes, Vol. 4, n°3-4, Automne-Hiver 1978.

[THE 80] THEVENOD-FOSSE P., DAVID R.,
"Un outil pour l'étude du test aléatoire de la partie opérative de microprocesseurs"
Colloque International sur la Fiabilité et la Maintenabilité,
Perros-Guirec/Trégastel, France, Sept. 1980.

[VEL 80a] ROBACH Ch., SAUCIER G., VELAZCO R.,
"Flexible test method for microprocessors"
Euromicro Symposium, Londres, Sept. 1980.

[VEL 80b] ROBACH Ch., SAUCIER G., VELAZCO R.,
"Le test fonctionnel paramètrable de microprocesseurs"
Revue RAIRO-Automatique, Vol. 14, n°3, Oct. 1980.

[VEL 81] SAUCIER G., VELAZCO R.,
"Microprocessor functional test using deterministic test pattern"
Euromicro Symposium, Paris, Sept. 1981.

[VEL 82a] BELLON C., LIOTHIN A., SADIER S., VELAZCO R.,
"Génération automatique de programmes de test par microprocesseurs et leur circuits annexes"
Rapport de Contrat IMAG-EMD, Feb. 1982.

[VEL 82b] BELLON C., LIOTHIN A., SADIER S., VELAZCO R.,
"Automatic test generation for microprocessors"
Design Automation Conference (à paraître) Juin 1982.

[WAC 80] WACKS K.P. and all,
"An integrated system for LSI device modeling"
IEEE Test Conference, USA, Cherry Hill, Nov. 1980.

ANNEXE 1

RECHERCHE DES OPÉRANDES DE TEST POUR LES OPÉRATIONS ARITHMETIQUES ET LOGIQUES

Les unités de traitement d'un microprocesseur (unité arithmétique et logique, circuits de décalage, compteurs,...) ne sont pas vérifiées en tant qu'éléments matériels mais par le bon déroulement de leurs actions, c'est-à-dire le bon fonctionnement de l'ensemble des micro-opérations. Ceci est d'autant plus intéressant que certaines micro-opérations impliquent l'activation de plusieurs unités de traitement.

α) *représentation des micro-opérations*

Les opérations réalisées par les unités de traitement d'un microprocesseur calculent un résultat Z à partir d'un ou plusieurs opérandes : X,Y,... Les opérandes et le résultat peuvent s'écrire comme das quantités booléennes générales dont le nombre de composantes est fonction du type d'opération et du microprocesseur considéré.

Une opération quelconque peut donc être représentée par une fonction booléenne générale de variables générales [15].

$Z = F(X,Y,...)$ où $X = x_1 \ldots x_n$

$Y = y_1 \ldots y_n$

$Z = z_1 \ldots z_p$, $p \geq n$

. Ecriture sous forme de fonctions simples

Toute fonction générale peut être considérée comme un système de p fonctions simples de variables générales :

$$Z = F(X,Y) \equiv \begin{cases} z_1 = f_1(X,Y) \\ \ldots\ldots\ldots\ldots \\ z_p = f_p(X,Y) \end{cases}$$

. Représentation des micro-opérations

On cherche à écrire les fonctions simples f_i de manière à mettre en évidence la dépendance entre la $i^{ème}$ composante du résultat et les $i^{èmes}$ composantes des opérandes.

Toute fonction booléenne f(X) peut être développée par rapport à l'une quelconque de ses variables simples.

$$f(X) = \overline{x_1} \cdot f(0,x_2,\ldots,x_n) + x_1 \cdot f(1, x_2,\ldots,x_n)$$

L'application successive de ce lemme à la fonction $f_i(X,Y,\ldots)$ permet d'obtenir un développement suivant les variables x_i, y_i,...

$$f_i(X,Y,\ldots) = \sum_j \tilde{x}_i \cdot \tilde{y}_i \cdot \ldots \cdot C_i^j$$

On distingue :

. <u>le corps</u> qui est le produit des variables $\tilde{x}_i$, $\tilde{y}_i$,... où $\tilde{x}_i = x_i$ ou $\overline{x_i}$;

. <u>le contexte</u> C_i^j qui est une fonction booléenne des variables x_k, y_k,... où $k \neq i$. La fonction de contexte est obtenue en remplaçant dans $f_i(X,Y,\ldots)$ les variables x_i, y_i,... du corps par 0 si ces variables sont complémentées dans le corps

1 sinon.

<u>Remarques</u> :

1. Par souci d'homogénéité on donnera à l'indice j de C_i^j la valeur décimale correspondant à la variable binaire $\tilde{x}_i \cdot \tilde{y}_i \ldots$ du corps
2. Pour certaines micro-opérations, le $i^{ème}$ bit du résultat z_i ne dépend que des $i^{èmes}$ bits des opérandes x_i, y_i,... . La fonction contexte est alors une constante (soit 0, soit 1).
3. Les composantes du résultat Z peuvent être décrites par une même fonction f_i : $\forall$ i, $f_i = f$.
4. Les fonctions du contexte peuvent être liées par une loi de récurrence.

<u>Exemples</u> :

1. Opérations à contexte constant et fonctions identiques :

   Complément $Z = \overline{X}$

   $$\forall\ i,\ z_i = \overline{x_i} \cdot 1 + x_i \cdot 0$$

   Opération ET : $Z = X \cdot Y$

   $$\forall\ i,\ z_i = x_i \cdot y_i \cdot 1 + \overline{x_i} \cdot y_i \cdot 0 + x_i \cdot \overline{y_i} \cdot 0 + \overline{x_i}\ \overline{y_i} \cdot 0$$

   $$C_i^0 = C_i^1 = C_i^2 = 0 \quad , \quad C_i^3 = 1$$

Opération OU : $Z = X + Y$

$$\forall\, i,\ z_i = \overline{x_i}\ \overline{y_i} \cdot 0 + \overline{x_i}\ y_i \cdot 1 + x_i\ \overline{y_i} \cdot 1 + x_i\ y_i \cdot 1$$

$$C_i^0 = 0\ ,\ C_i^1 = C_i^2 = C_i^3 = 1$$

2. Opérations à fonctions identiques

Complément à 2 : $Z = 2^n - |X|$

$$\forall\, i,\ z_i = x_i \cdot (x_{i-1}+\ldots+x_1) + \overline{x_i} \cdot (\overline{x_{i-1}} \cdot \ldots \cdot \overline{x_1})$$

$$C_i^0 = \overline{x_{i-1}} \cdot \ldots \cdot \overline{x_1}\ ,\ C_i^1 = x_{i-1} + \ldots + x_1$$

Décalage à gauche de 1 position :

$$\forall\, i,\ z_i = x_i \cdot x_{i-1} + \overline{x_i} \cdot x_{i-1}$$

3. Opération dont les fonctions contextes sont liées par une loi de récurrence :

Addition :

$$\forall\, i \in [1,n]\ ,\ z_i = \overline{x_i}\ \overline{y_i}\ C_i^0 + \overline{x_i}\ y_i\ C_i^1 + x_i\ \overline{y_i}\ C_i^2 + x_i\ y_i\ C_i^3$$

avec $C_i^0 = C_i^3 = C_i = x_{i-1} \cdot y_{i-1} + C_{i-1}(x_{i-1}+y_{i-1})$

$$C_i^1 = C_i^2 = \overline{C_i}$$

β) *Le test des microopérations.*

Le test des micro-opérations revient à déterminer un ensemble de valeurs pour les opérandes d'entrée X, Y,... suffisant pour assurer le bon fonctionnement de cette opération. Le test se fait par la vérification du système d'équations par lequel l'opération est décrite.

L'obtention des vecteurs de test pour une opération se fait en deux étapes:

* le test des f_i c'est-à-dire la détermination de l'ensemble de test E_i pour chaque fonction f_i ;
* la composition de ces ensembles E_i de manière à définir l'ensemble E des vecteurs de test de F.

· le test d'une fonction f_i

Toute fonction f_i peut s'écrire comme une somme booléenne :

$$f_i = \sum_j (\tilde{x}_i \cdot \tilde{y}_i \cdot \ldots) \cdot C_i^j$$

Il est évident que le test exhaustif d'une telle fonction n'est pas envisageable. La solution adoptée consiste à faire :

- un test exhaustif sur le corps c'est-à-dire à prendre toutes les combinaisons possibles des $i^{èmes}$ bits des opérandes,
- un test partiel sur le contexte, en affectant la valeur VRAI(1) et la valeur FAUX(0) à chaque contexte.

La fonction est alors vérifiée pour chacun de ses termes de la manière suivante : pour un terme $\tilde{x}_i \cdot \tilde{y}_i \cdot \ldots \cdot C_i^j$, on affecte aux variables $\tilde{x}_i, \tilde{y}_i, \ldots$ les valeurs booléennes correspondantes (1 si $\tilde{x}_i = x_i$, 0 si $\tilde{x}_i = \bar{x}_i$) et aux contextes les valeurs VRAI et FAUX.

Remarques :

. Les valeurs des variables de la fonction de contexte sont choisies arbitrairement si aucun renseignement précis sur l'algorithme de calcul de la micro-opération n'est connu ;

. Le nombre de vecteurs de test d'une fonction f_i peut être déterminé directement par le nombre d'opérandes de la micro-opération et la nature de la fonction contexte (constante ou non).

Exemple : - Le nombre de vecteurs de test est égal à 2^{m+1} pour une fonction à m opérandes et contextes non constants ;
- Il est égal à 2^m pour une fonction à m opérandes et contextes constants.

Pour chaque terme d'une fonction f_i à contexte non constant on définit donc deux vecteurs de test :

V_0^j pour $C_i^j = 0$

V_1^j pour $C_i^j = 1$

Exemples :

. Complément $Z = \overline{X}$

$$\forall\ i,\ z_i = x_i \cdot 1 + \overline{x}_i \cdot 0$$

les fonctions contexte étant constantes, on obtient deux vecteurs de test :

	x_1 ... x_{i-1}	x_i	x_{i+1} ... x_n
V^1	ϕ ... ϕ	1	ϕ ... ϕ
V^0	ϕ ... ϕ	0	ϕ ... ϕ

. Opération logique quelconque à 2 opérandes

$$\forall\ i,\ z_i = \overline{x}_i\ \overline{y}_i\ C_i^0 + \overline{x}_i\ y_i\ C_i^1 + x_i\ \overline{y}_i\ C_i^2 + x_i\ y_i\ C_i^3$$

Les fonctions contexte étant constantes on obtient donc quatre vecteurs de test :

	x_1 ... x_{i-1}	x_i	x_{i+1} ... x_n ; y_1 ... y_{i-1}	y_i	y_{i+1} ... y_n
V^0	ϕ ... ϕ	0	ϕ ... ϕ ; ϕ ... ϕ	0	ϕ ... ϕ
V^1	ϕ ... ϕ	0	ϕ ... ϕ ; ϕ ... ϕ	1	ϕ ... ϕ
V^2	ϕ ... ϕ	1	ϕ ... ϕ ; ϕ ... ϕ	0	ϕ ... ϕ
V^3	ϕ ... ϕ	1	ϕ ... ϕ ; ϕ ... ϕ	1	ϕ ... ϕ

. Addition : Z = X ADD Y

$$\forall\ i \in [1,n],\ z_i = \overline{x_i}\ \overline{y_i}\ C_i^0 + \overline{x_i}\ y_i\ C_i^1 + x_i\ \overline{y_i}\ C_i^2 + x_i\ y_i\ C_i^3$$

$$\text{avec } C_i^0 = C_i^3 = C_i = x_{i-1}\ y_{i-1} + C_{i-1}(x_{i-1}+y_{i-1})$$

$$C_i^1 = C_i^2 = \overline{C}_i$$

$$C_0 = 0$$

Choix des valeurs du contexte :

$$C_i = 0 \quad \Rightarrow \quad \begin{cases} x_{i-1} = y_{i-1} = 0 \\ \quad \text{ou} \\ x_{i-1} \cdot y_{i-1} = 0 \quad \text{et} \quad C_{i-1} = 0 \end{cases}$$

$$C_i = 1 \quad \Longrightarrow \quad \begin{cases} x_{i-1} = y_{i-1} = 1 \\ \text{ou} \\ y_{i-1} = C_{i-1} = 1 \\ \text{ou} \\ x_{i-1} = C_{i-1} = 1 \end{cases}$$

On choisit une valeur du contexte telle qu'il y ait le maximum de variables indéterminées ($x_k = \phi$). Un ensemble E_i de vecteurs peut alors être le suivant :

	x_1	...	x_{i-2}	x_{i-1}	x_i	x_{i+1}	...	x_n ;	y_1	...	y_{i-2}	y_{i-1}	y_i	y_{i+1}	...	y_n
V_0^0	ϕ	...	ϕ	0	0	ϕ	...	ϕ ;	ϕ	...	ϕ	0	0	ϕ	...	ϕ
V_1^0	ϕ	...	ϕ	1	0	ϕ	...	ϕ ;	ϕ	...	ϕ	1	0	ϕ	...	ϕ
V_0^1	ϕ	...	ϕ	1	0	ϕ	...	ϕ ;	ϕ	...	ϕ	1	1	ϕ	...	ϕ
V_1^1	ϕ	...	ϕ	0	0	ϕ	...	ϕ ;	ϕ	...	ϕ	0	1	ϕ	...	ϕ
V_0^2	ϕ	...	ϕ	1	1	ϕ	...	ϕ ;	ϕ	...	ϕ	1	0	ϕ	...	ϕ
V_1^2	ϕ	...	ϕ	0	1	ϕ	...	ϕ ;	ϕ	...	ϕ	0	0	ϕ	...	ϕ
V_0^3	ϕ	...	ϕ	0	1	ϕ	...	ϕ ;	ϕ	...	ϕ	0	1	ϕ	...	ϕ
V_1^3	ϕ	...	ϕ	1	1	ϕ	...	ϕ ;	ϕ	...	ϕ	1	1	ϕ	...	ϕ

Note : Les vecteurs de test sont notés de telle sorte que :

V_i^j

i indique la valeur du contexte

i = 0 contexte FAUX

i = 1 contexte VRAI

j indique la valeur binaire du corps

j = 0 correspond à $\overline{x_i}\ \overline{y_i} \ldots C^0$

j = 1 - à $\overline{x_i}\ y_i \ldots C^1$

j = 2 - à $x_i\ \overline{y_i} \ldots C^2$

j = 3 - à $x_i\ y_i \ldots C^3$

. Composition des ensembles de test E_i

A chaque fonction f_i est associé un ensemble de vecteurs de test E_i constitué de vecteurs ϕ- booléens.

L'ensemble de test E de la fonction F est donc l'ensemble

$$\{E_i\}_{i \in [1,p]}$$

A partir de l'ensemble $\{E_1,\ldots,E_p\}$ on cherche à déterminer l'ensemble pseudo-minimal E^* tel que $\forall\, i \in [1,p]$, $E_i \in E^*$, en affectant une valeur aux variables indéterminées.

. Relation de compatibilité

Deux quantités booléennes générales A et B sont dites compatibles s'il n'existe aucun indice i tel que ($x_i = 0$ et $y_i = 1$) ou ($x_i = 1$ et $y_i = 0$).

On note cette relation $A \sim B$.

Les jeux de valeurs possibles sont donc :

(0,0)

(0,ϕ)

(ϕ,ϕ)

(1,ϕ)

(1,1)

. Détermination de l'ensemble pseudo-minimal E^*

Ceci revient au problème suivant : étant donné un ensemble S de vecteurs ϕ- booléens, déterminer l'ensemble S^* tel que :

1. $\forall\, s \in S$, $\exists\, s^* \in S^*$ tel que $s \sim s^*$
2. S^* est de cardinalité minimale.

On appelle S^* ensemble compatible minimal.

. Algorithme de recherche de E^* :

L'algorithme est appliqué à l'ensemble :

$E = \{E_1,\ldots,E_p\}$ de vecteurs ϕ- booléens à n composantes dans le but d'obtenir un ensemble compatible pseudo-minimal E^*.

Principe : 1. On détermine de manière constructive des sous-ensemble A_i de E tels que :

$$\forall A, \forall B \in A_i, A \sim B$$

les vecteurs de A_i sont compatibles deux à deux.

Remarque : La relation de compatibilité restreinte aux éléments de A_i est une relation d'équivalence :

$$\forall A, \forall B, \forall C \in A_i \begin{cases} A \sim A & \text{par définition de } \sim \\ A \sim B \Rightarrow B \sim A & \\ \left.\begin{matrix} A \sim B \\ B \sim C \end{matrix}\right\} \Rightarrow A \sim C & \text{par construction de } \bar{A}_i \end{cases}$$

2. On choisit comme représentant d'un sous-ensemble A_i le vecteur ϕ- booléen

$S_i = (s_1 \ldots s_n)$ défini comme suit :

$$\begin{cases} s_k = 0 & \text{si } \exists A \in A_i \; ; \; a_k = 0 \\ s_k = 1 & \text{si } \exists B \in A_i \; ; \; b_k = 1 \\ s_k = \phi & \text{si } \forall C \in A_i \; ; \; c_k = \phi \end{cases}$$

S_i est appelé signature de A_i.

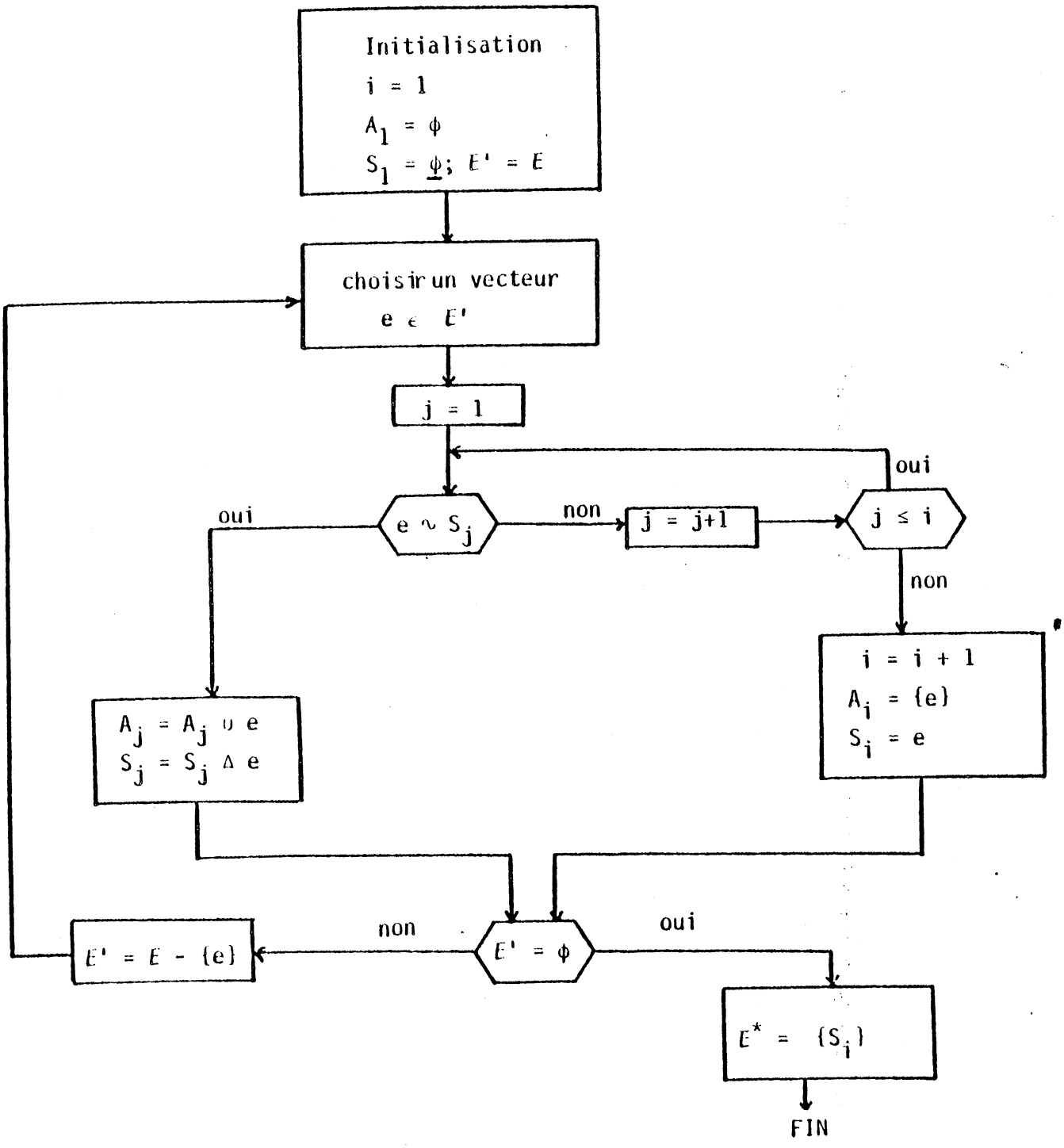
L'ensemble des signatures constitue l'ensemble de test de la fonction F

$$E^* = \{S_i\}$$

Remarques : . S_i est compatible avec tout élément de A_i

. un vecteur ϕ-booléen quelconque W compatible avec S_i appartient à A_i.

algorithme :



Les vecteurs de test obtenus S_i sont des vecteurs ϕ- booléens : le choix des valeurs indéterminées est alors indifférent.

. ϕ est l'ensemble vide

. $\underline{\phi}$ est le vecteur booléen dont toutes les composantes sont indéterminées:

$$\underline{\phi} = (\phi \ldots \phi)$$

. L'opération Δ est définie par le tableau de vérité suivant :

	0	1	ϕ
0	0	-	0
1	-	1	1
ϕ	0	1	ϕ

Cette opération étant appliquée sur des vecteurs compatibles on ne peut avoir ni $0\Delta 1$ ni $1\Delta 0$.

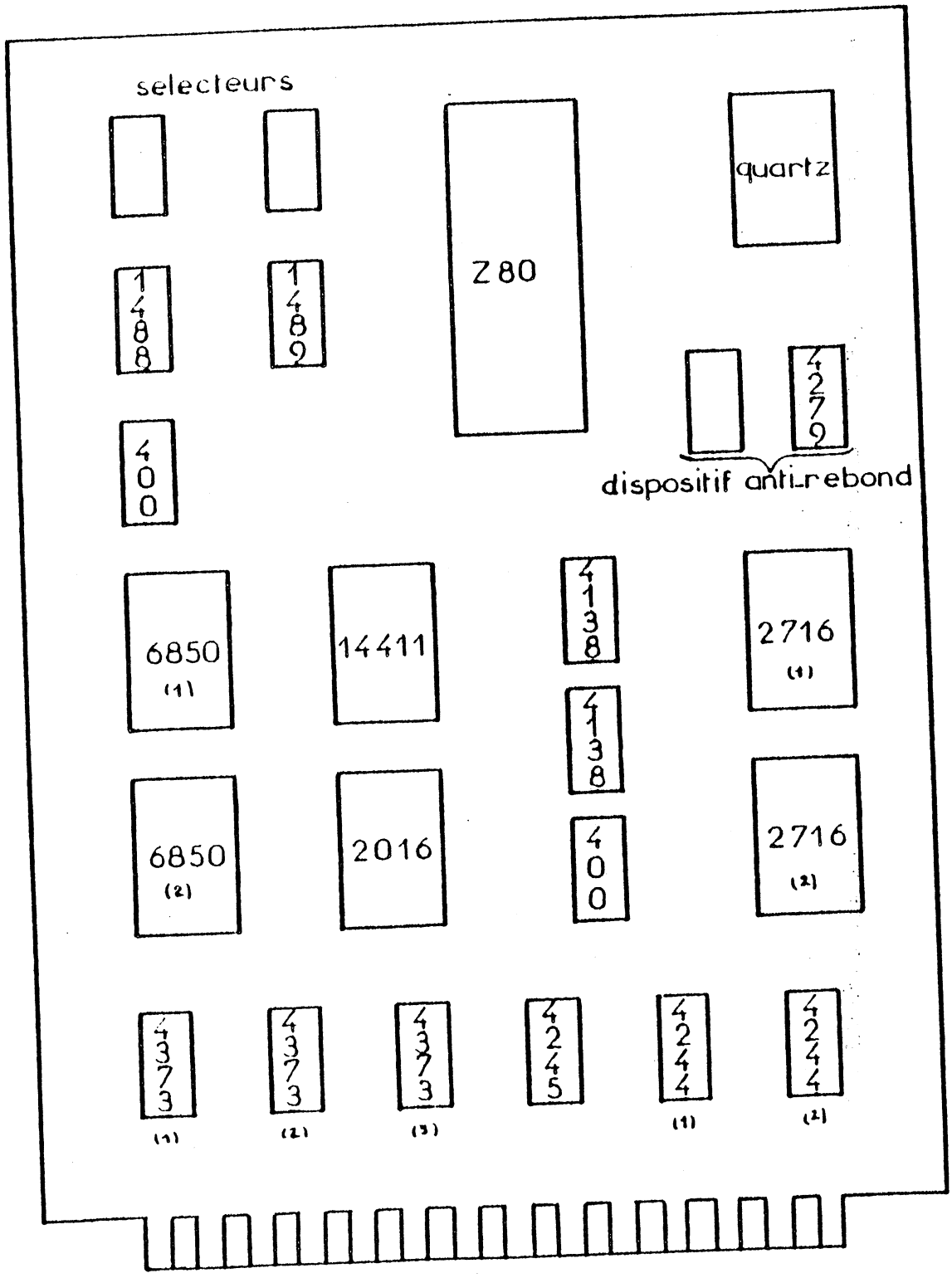
Remarque : L'algorithme proposé fournit un ensemble de test non minimal , la minimalité étant fonction de la manière dont sont choisis les vecteurs e.

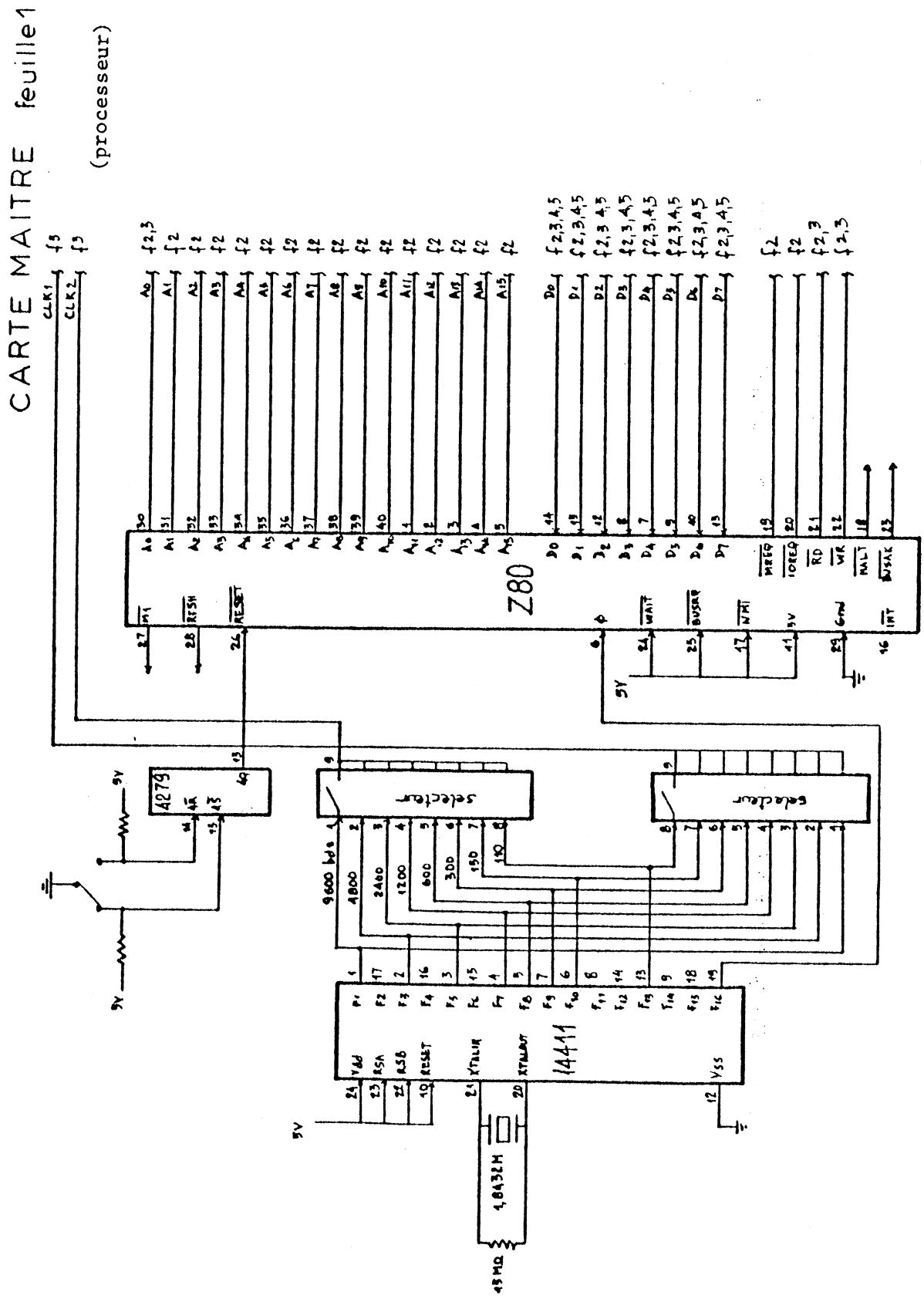
ANNEXE 2

SCHEMAS DÉTAILLÉS DU TESTEUR

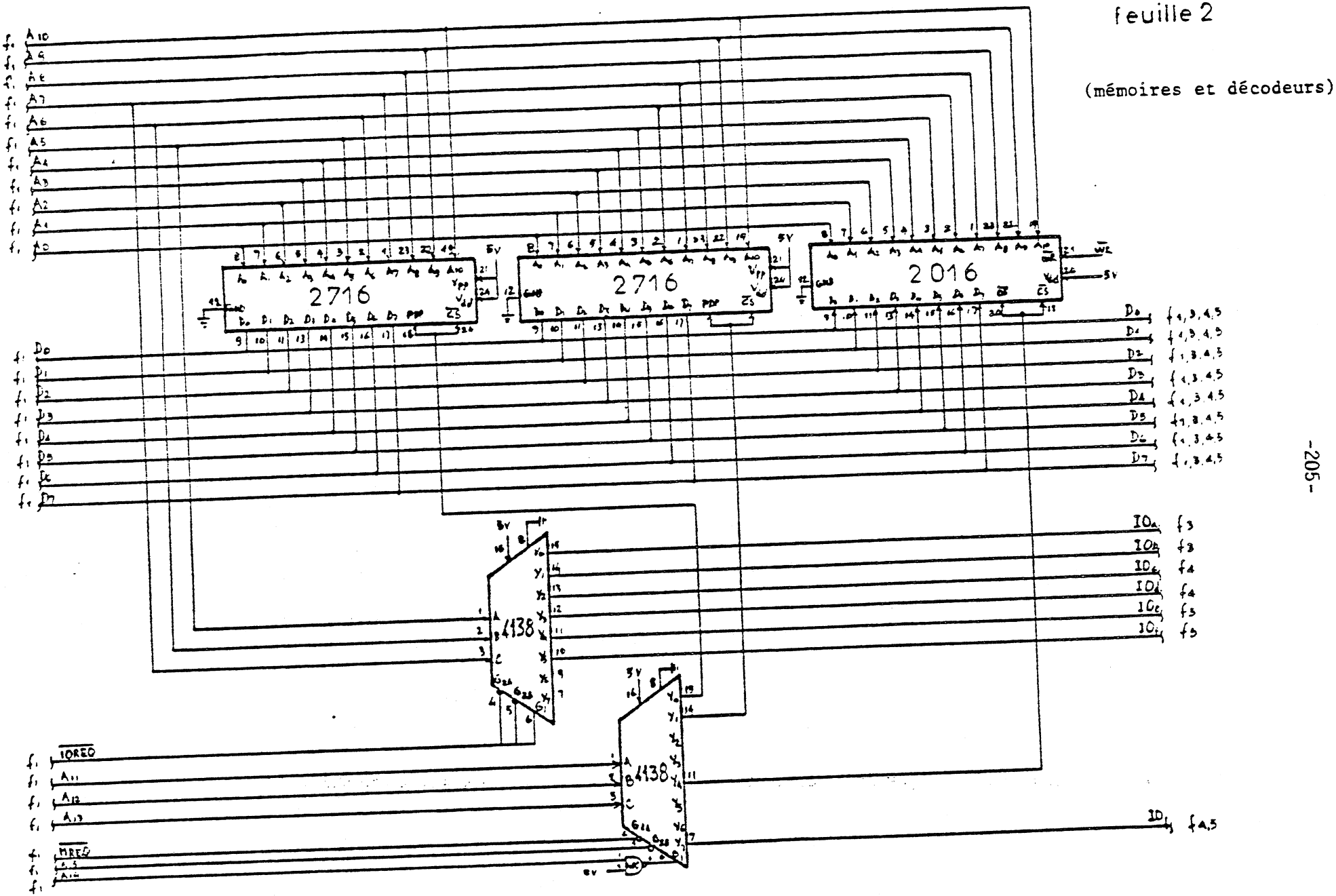
Cette annexe contient tous les schémas détaillés du testeur, la description du bus de fond de panier et un récapitulatif des différents espaces d'adressage de la carte maître.

CARTE MAITRE



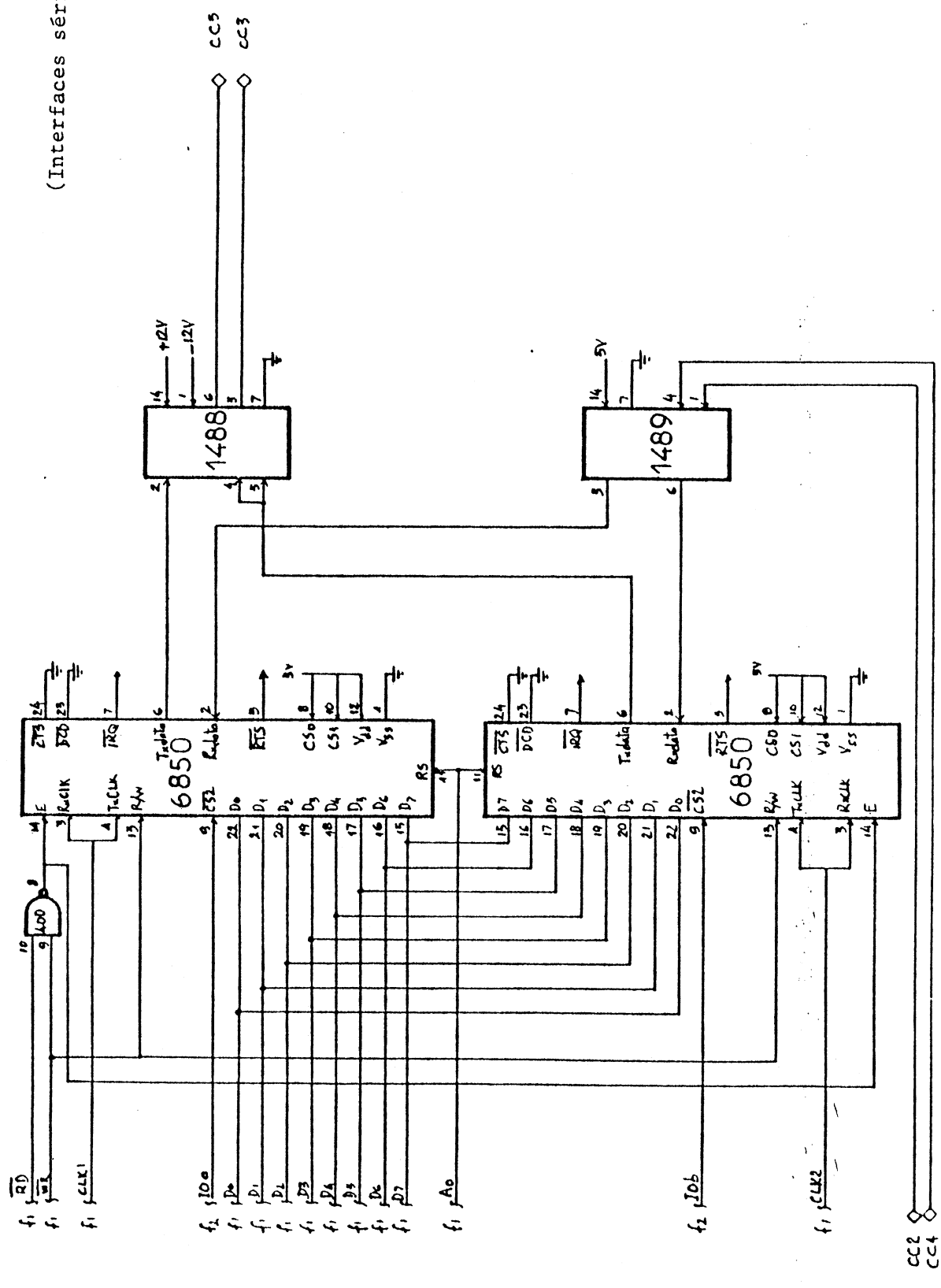

CARTE MAITRE feuille1
(processeur)
Z80
14411
4279
1,8432M
43 MΩ
5V
9600 bps
4800
2400
1200
600
300
150
110

(mémoires et décodeurs)



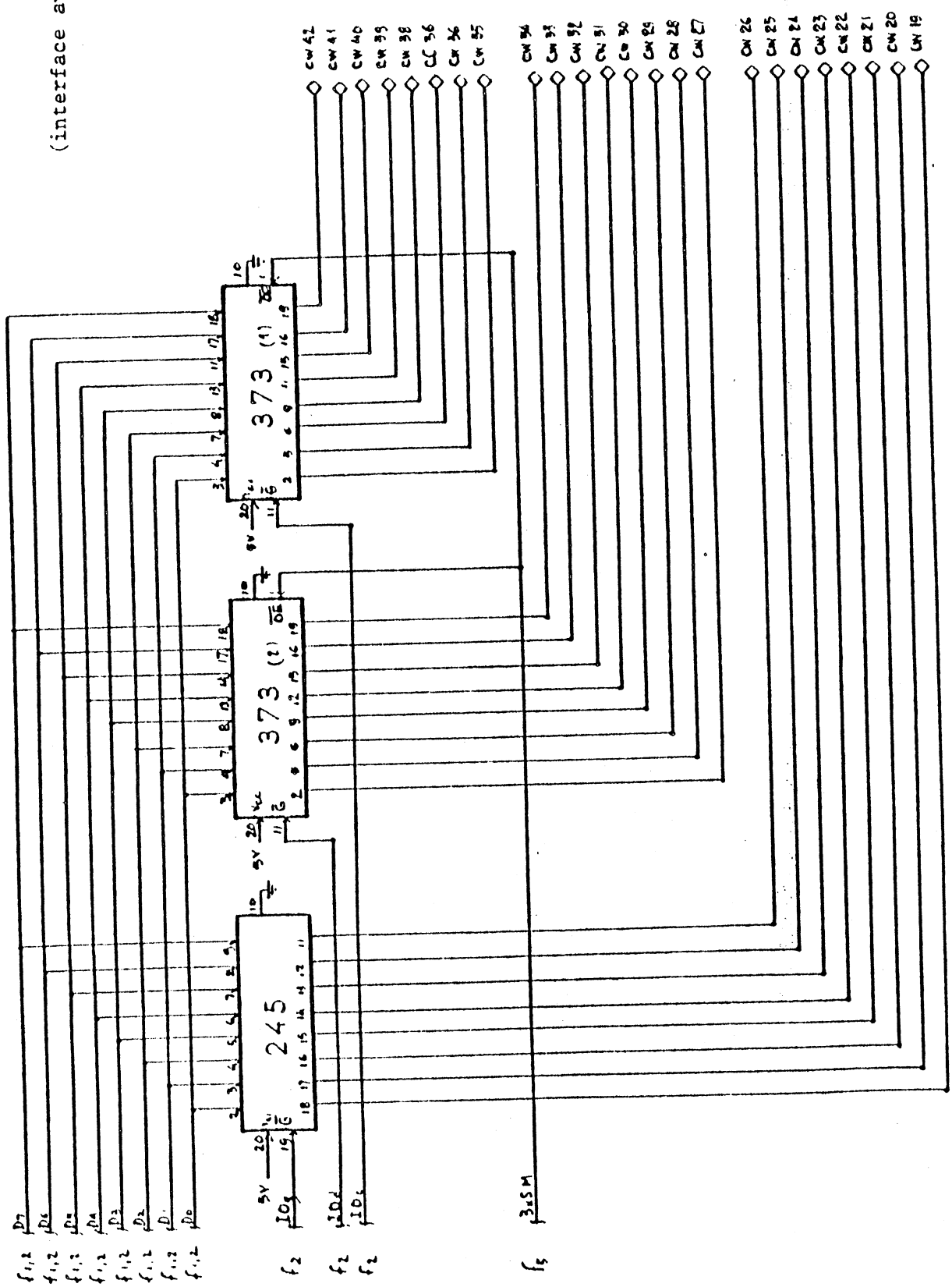
feuille 3

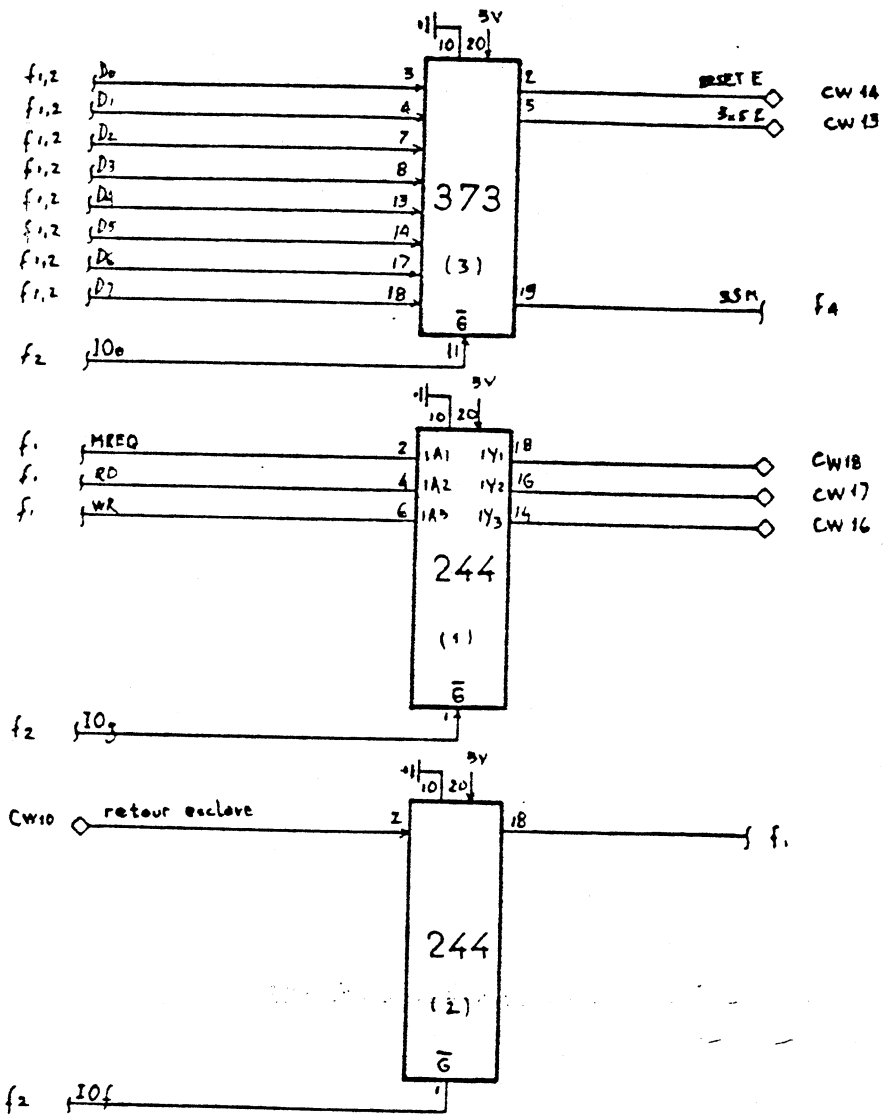
(Interfaces séries)



feuille 4

(interface avec bus externe)

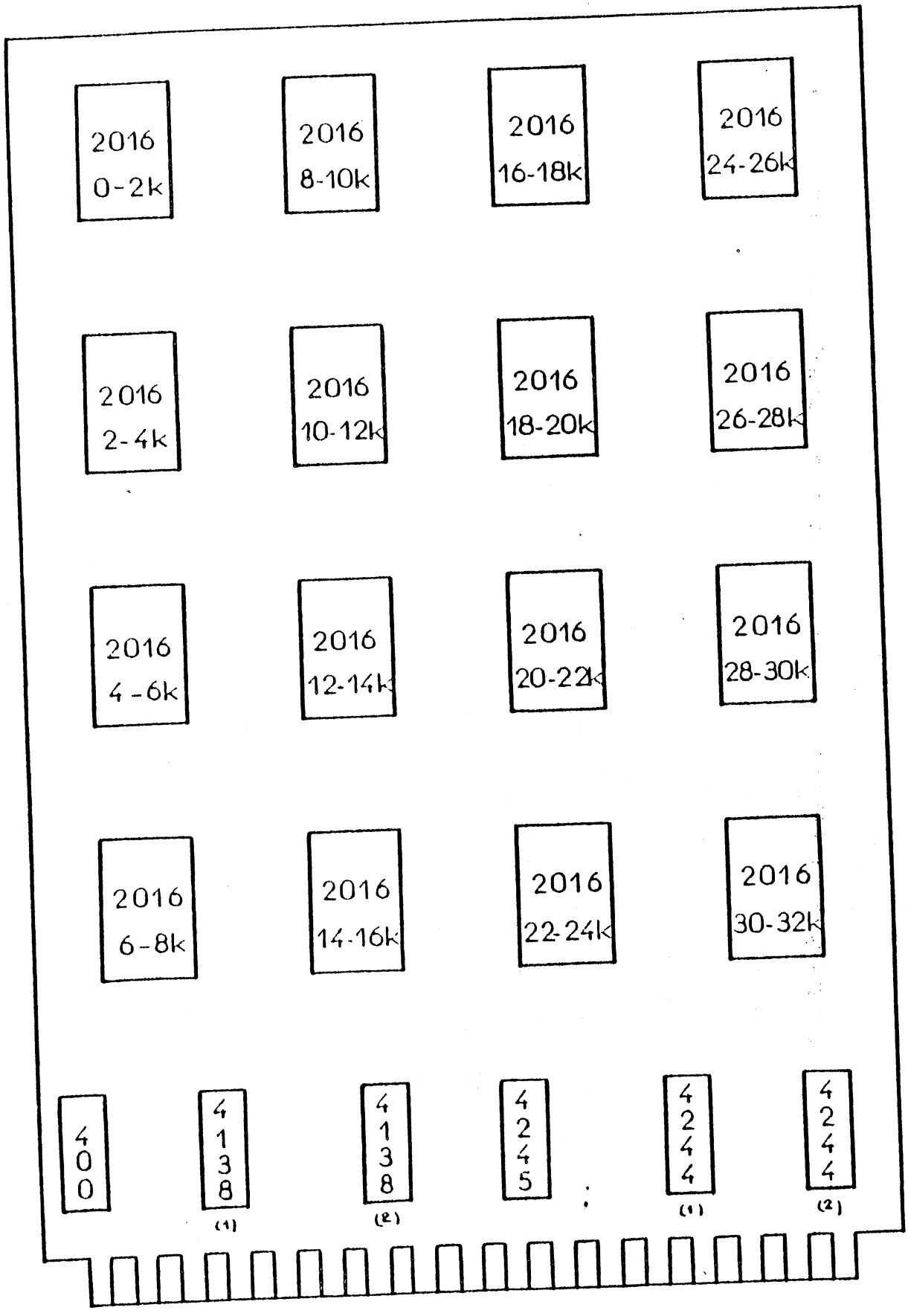




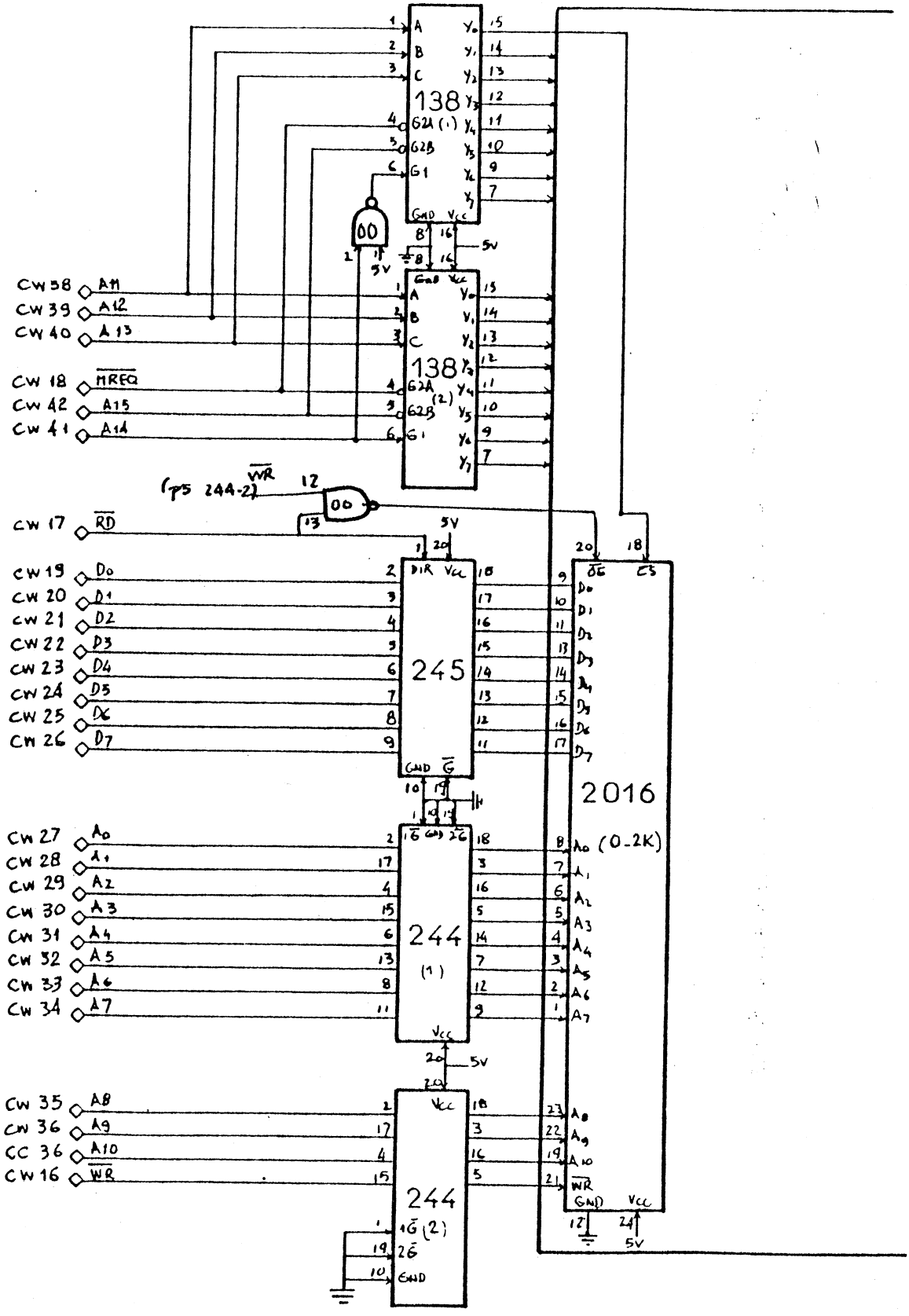
feuille 5

(Interface avec bus externe)

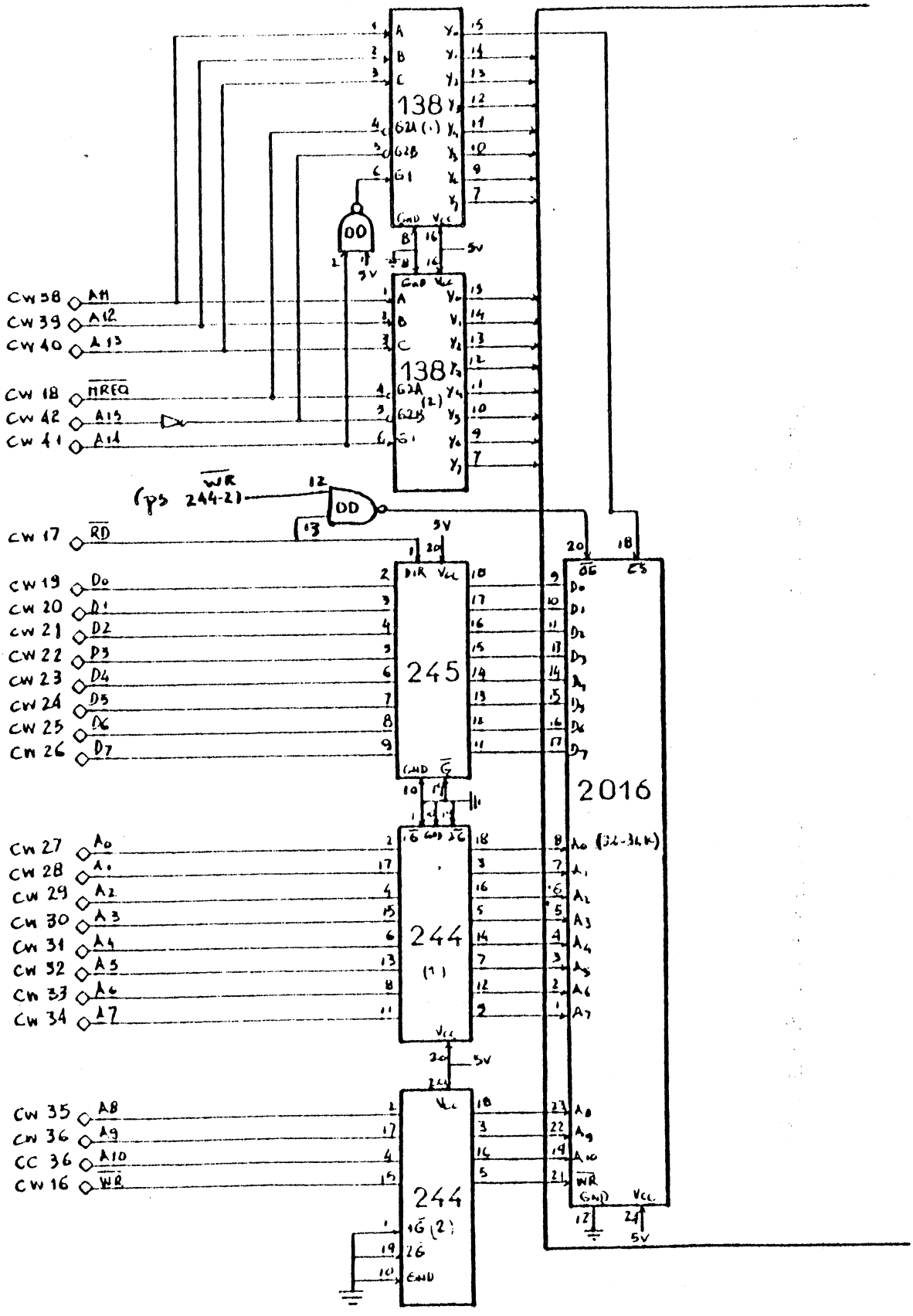
CARTE MEMOIRE 0-32k



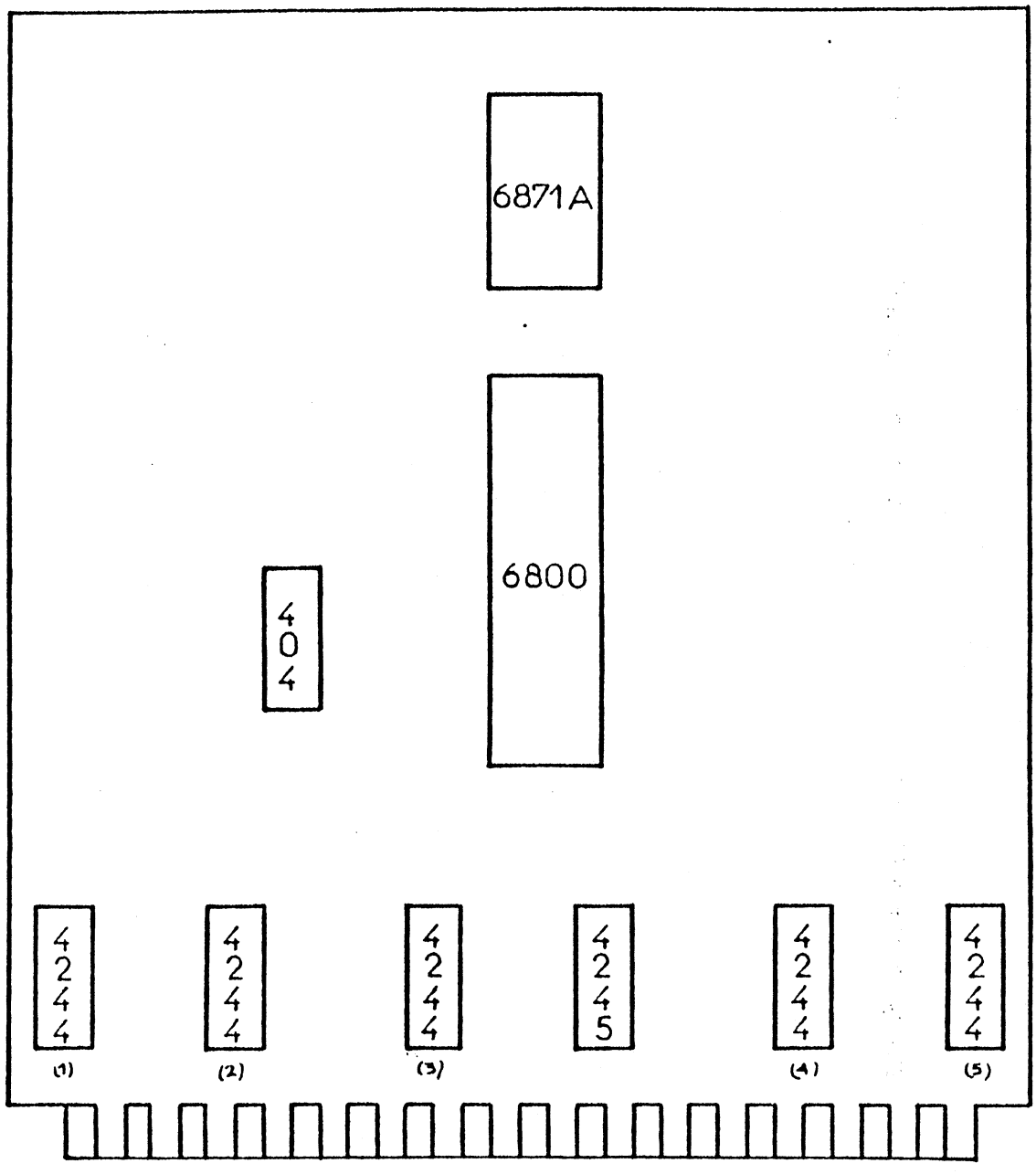
CARTE MEMOIRE (0 - 32K)



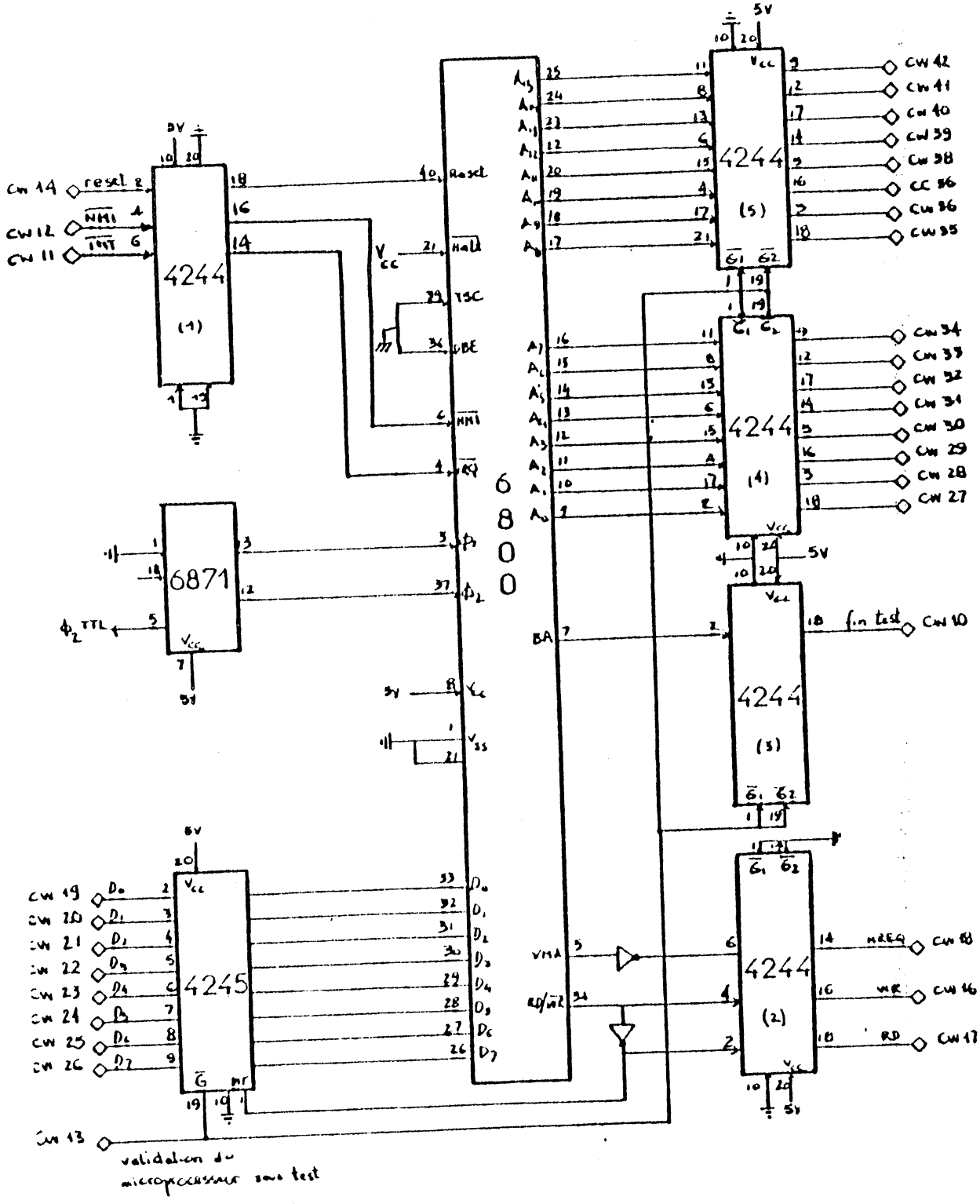
CARTE MEMOIRE (32 - 64K)



CARTE ESCLAVE



CARTE ESCLAVE



CW = Côté Wrapping
CC = Côté Composant

DESCRIPTION DU BUS DE FOND DE PANIER

Côté composants				Côté wrapping	
	GND	1		GND	
(ACIA1)	RX	2			
"	TX	3			
(ACIA2)	RX	4			
"	TX	5			
(bouton reset)	RESET 1	6			
" "	RESET 2	7			réception des signaux du microprocesseur sous test
		8			
		9			
		10		Fin du test	
		11		$\overline{INT}$	
		12		$\overline{NMI}$	
		13		3-state microprocesseur sous test	
		14		Reset du microprocesseur sous test (low)	
		15			
		16	$\overline{WR}$		CONTROLE
		17	$\overline{RD}$		
		18	$\overline{MREQ}$		
		19	D0		DONNEES
		20	D1		
		21	D2		
		22	D3		
		23	D4		
		24	D5		
		25	D6		
		26	D7		
		27	A0		ADRESSES
		28	A1		
		29	A2		
		30	A3		
		31	A4		
		32	A5		
		33	A6		
		34	A7		
		35	A8		
	A 10	36	A9		
	détrompeur	37	détrompeur		
		38	A11		
		39	A12		
		40	A13		
		41	A14		
		42	A15		
	VCC	43	VCC		

ESPACE ADRESSAGE DE LA CARTE MAITRE

MREQ

0 - 7FF	:	2716	(0 → 3FF : moniteur)
800 - FFF	:	2716	
2000 - 27FF	:	2016	(TOSHIBA) (2000 → 2072 : moniteur)
3800	:	Validation fond de panier	

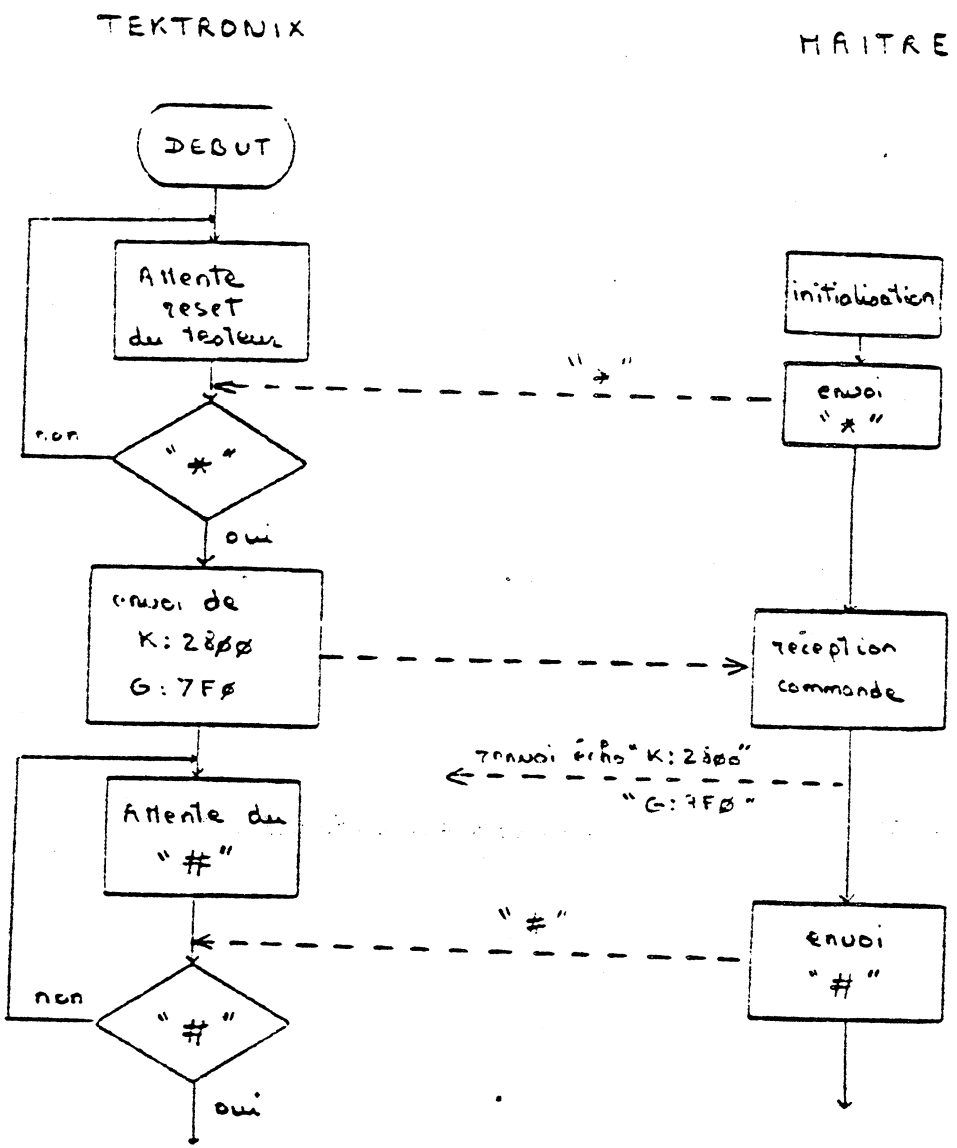
IOREQ

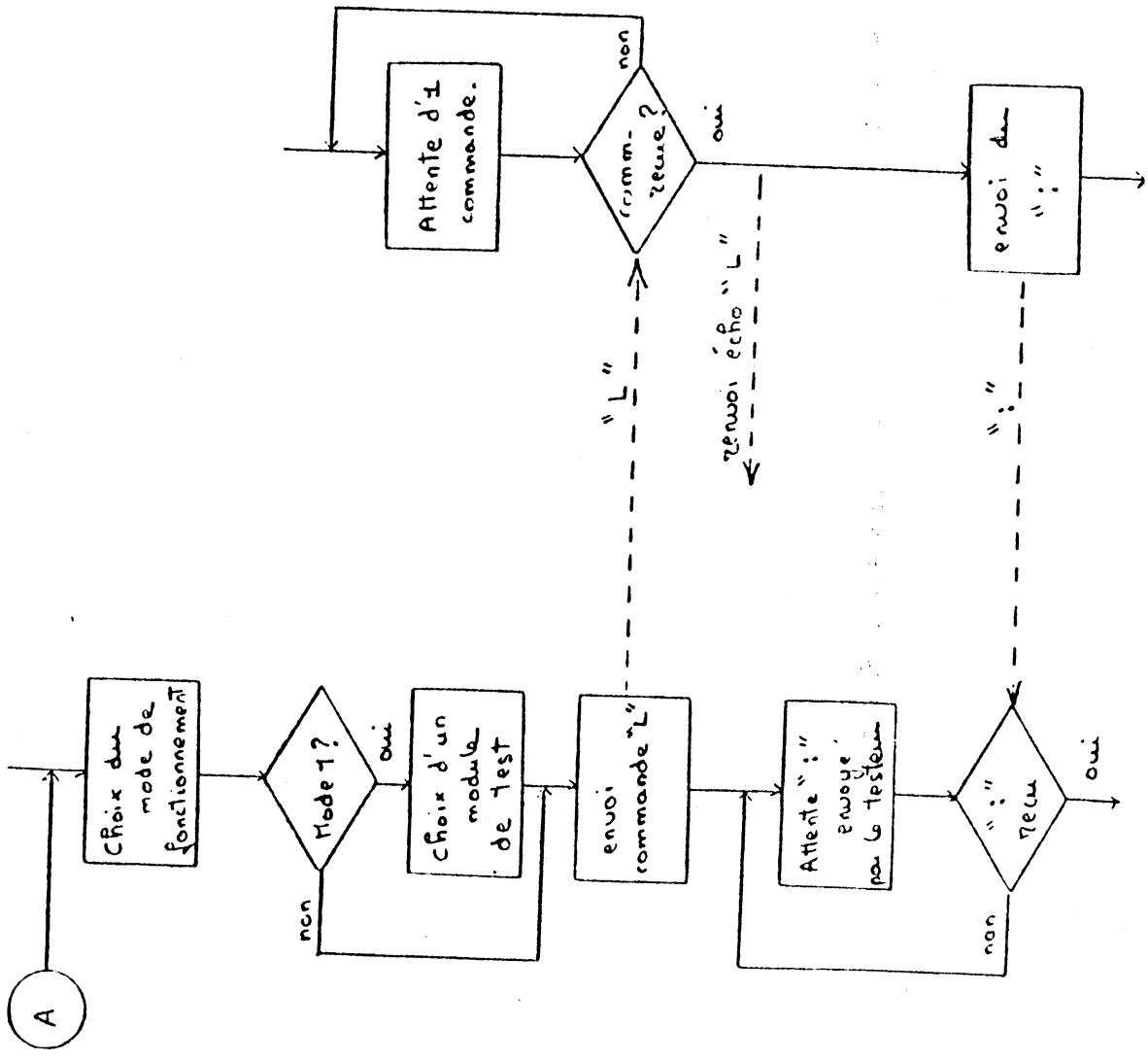
00	:	ACIA1 contrôle
01	:	" données
20	:	ACIA2 contrôle
		" données
40	:	adresse poids fort } fond de panier
60	:	adresse poids faible }
80	:	commandes vers microprocesseur sous test
A0	:	réception des signaux du microprocesseur sous test

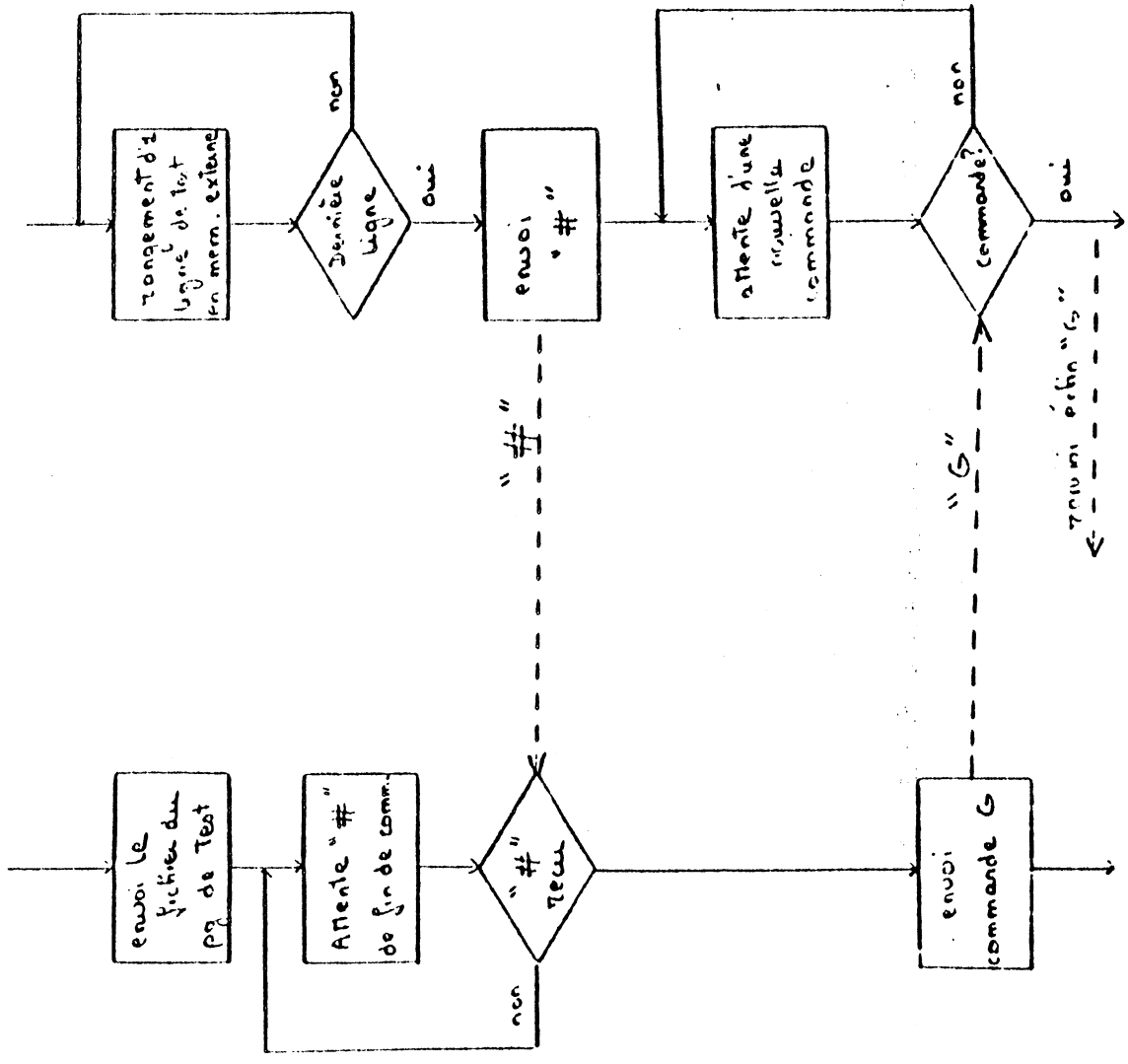
ANNEXE 3

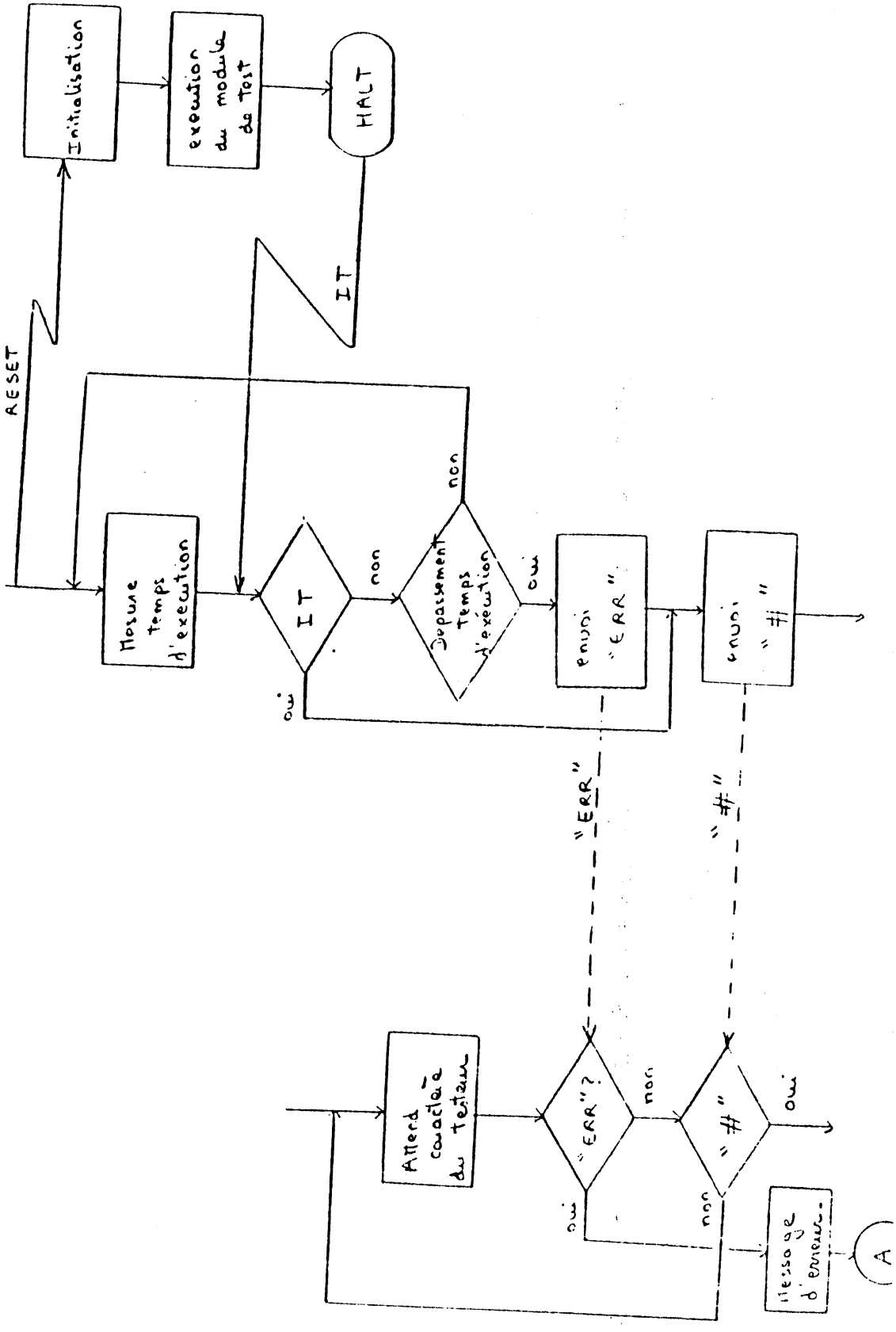
ORGANIGRAMMES DES DIFFÉRENTS MODULES DU MONITEUR

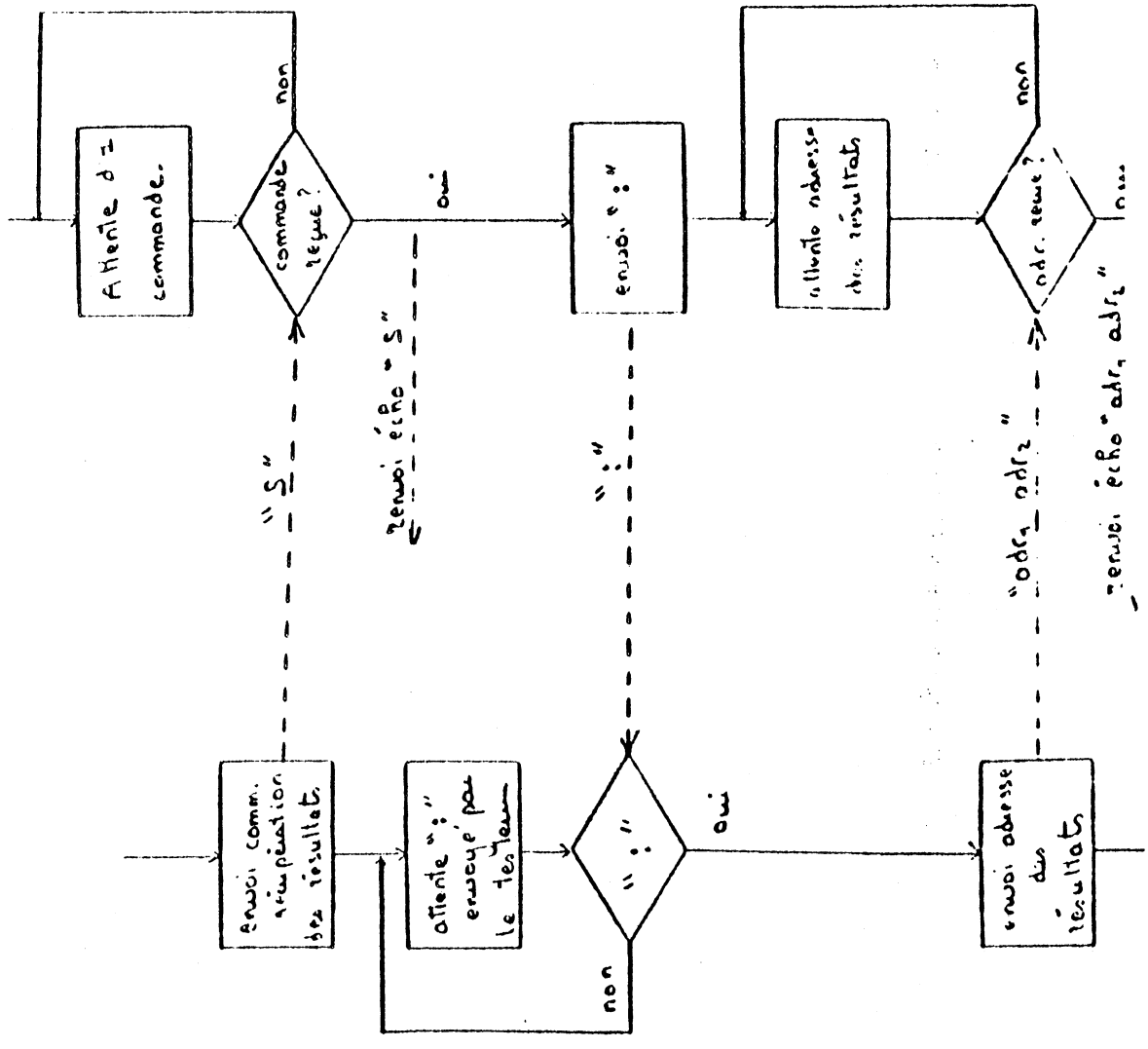
L'ordonnancement des opérations lors de l'exécution d'un module de test est illustré par les organigrammes suivants dans lesquels est mise en évidence la synchronisation entre le Tektronix, la carte maître du testeur et le microprocesseur sous test.

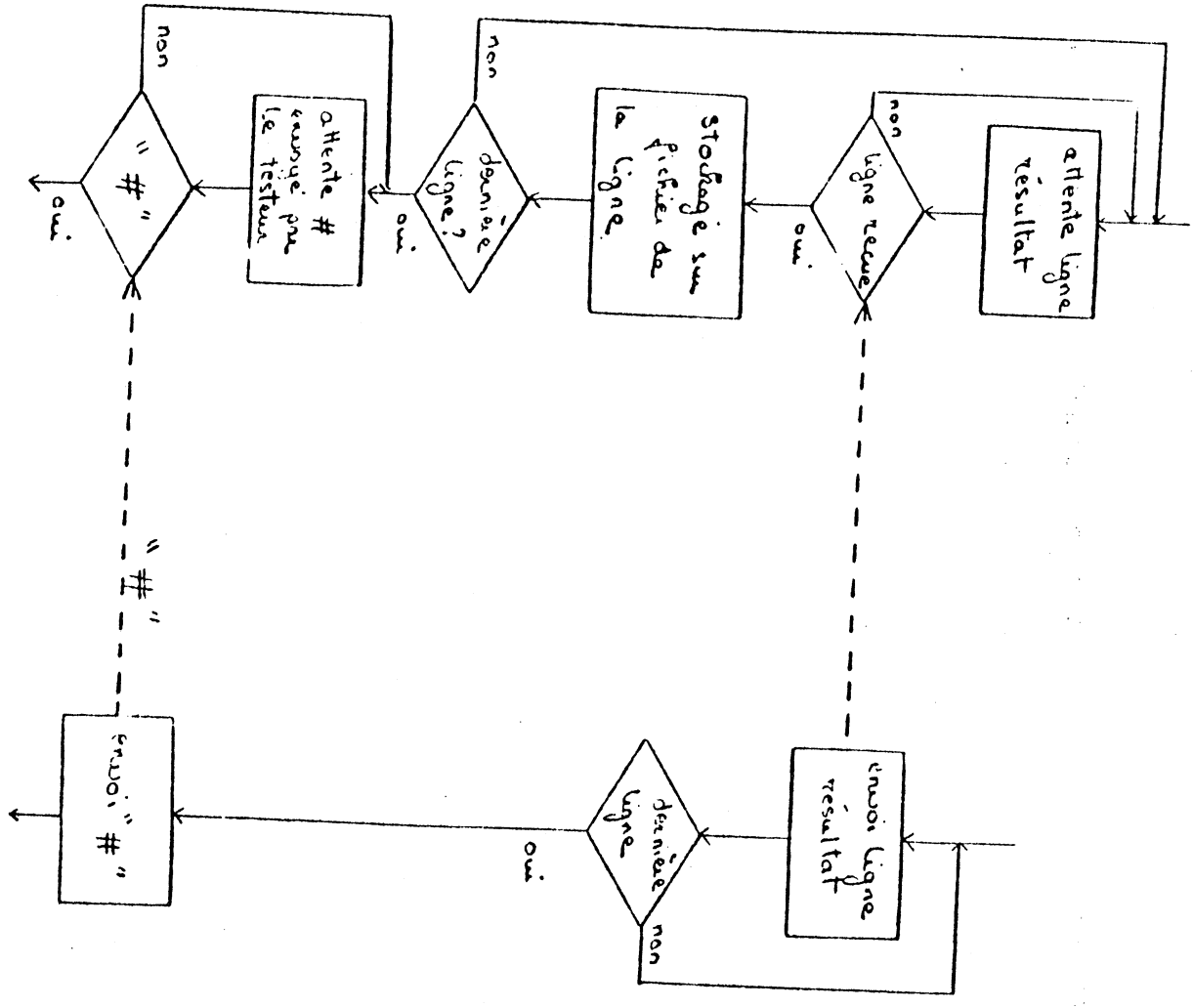

TEKTRONIX
MAITRE
ESCLAVE
DEBUT
Attente reset du traceur
initialisation
envoi "*"
"*"
non
"*"
oui
envoi de K:28ØØ G:7FØ
reception commande
renvoi écho "K:28ØØ"
"G:7FØ"
Attente du "#"
"#"
envoi "#"
non
"#"
oui

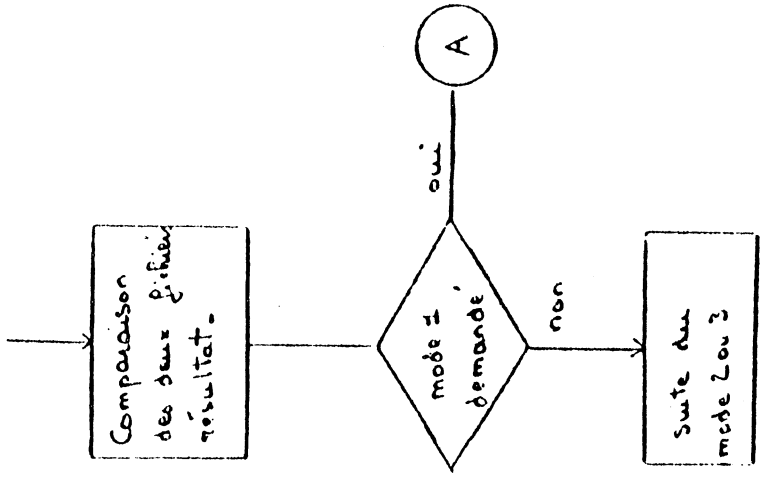

A
Choix du mode de fonctionnement
Mode 1 ?
oui
non
Choix d'un module de test
envoi commande "L"
Attente ":" envoyé par le testeur
":" reçu
oui
non
"L"
renvoi écho "L"
Attente d'1 commande.
reconnue ?
non
oui
envoi du ":"
":"


envoi le fichier du prg de Test
Attente "#" de fin de comm.
"#" reçu
non
envoi commande G
"G"
commande?
non
oui
attente d'une nouvelle commande
envoi "#"
dernière ligne
non
oui
rangement de la ligne de Test en mem. externe
retour début "G"


Initialisation
exécution du module de test
HALT
IT
RESET
Mesure temps d'exécution
IT
non
oui
Dépassement temps d'exécution
non
oui
envoi "ERR"
envoi "#"
"ERR"
"#"
Attend caractère du testeur
"ERR"?
non
oui
"#"
oui
non
Message d'erreur
A


Attente d'=
commande.
commande reçue ?
non
oui
envoi "="
envoi écho "="
envoi ":"
attente résultats
adr. reçue ?
non
oui
envoi comm. + réception résultats
attente ":" envoyé par le testeur
":" ?
non
oui
envoi adresse des résultats
"adr1 adr2"
envoi écho "adr1 adr2"


attente ligne résultat
ligne reçue
non
oui
stockage en fichier de la ligne
dernière ligne ?
non
oui
attente # envoyé par le testeur
"#"
non
oui
envoi ligne résultat
dernière ligne
non
oui
envoi "#"
"#"


Comparaison des deux fichiers résultat.
mode 1 demandé
oui
non
A
Suite Aux mode 2 ou 3

ANNEXE 4

REPRESENTATION DU JEU D'INSTRUCTIONS DU 6800 PAR LES GRAPHES D'EXÉCUTION ABSTRAITE

MESURE D'ACCESSIBILITE

- mémoire et extérieur — $t = t' = 1$
- registres A, B ; SP, PC, IX — $t = t' = 2$
- flag F — $t = t' = 3$

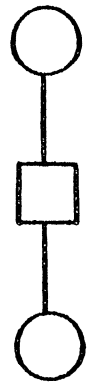
remarque : le registre de code condition est chargé et observé à partir de A

- remarque : SP pointe sur la première zone vide de la pile

ex. : PUSH A := A → Mem (SP)
SP ← SP - 1

CLASSE 1

S-classe : a

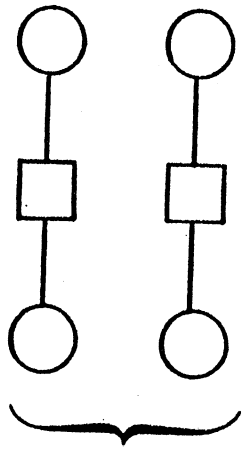


a

r')	OP	FORMAT 8 BITS	COMMENTAIRES	OP	FORMAT 16 BITS	COMMENTAIRES
?)		CLRA CLRB LDAA # n LDAB # n			LDX # m LDS # m JMP # m	
?)		TAB TBA				
	*	DAA				
	*	INCA		*	INS	
	*	INCB		*	INX	
	*	DECA		*	DES	
	*	DECB		*	DEX	
	*	NEGA				
	*	NEGB				
	*	ASLA LSRA NOP				
	*	ASLB LSRB				
	*	ASRA				
	*	ASRB				
	*	COMA				
	*	COMB				
3)		TAP TPA				
		TSTA TSTB				
3)	*	CLC				
	*	CLI				
	*	CLV				
	*	SEC				
	*	SEI				
	*	SEV				

CLASSE 2

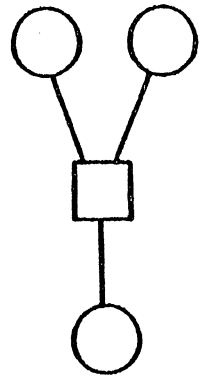
S-classe : b



(T,T')	OP	FORMAT 8 BITS	COMMENTAIRES	OP	FORMAT 16 BITS	COMMENTAIRES
(2,2)				*	TXS	
				*	TSX	

CLASSE 2

S-classe : a)

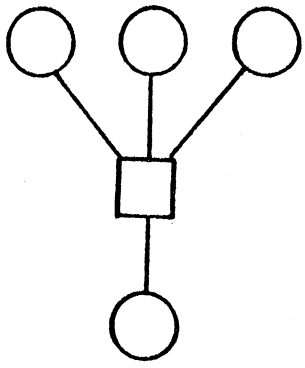


a

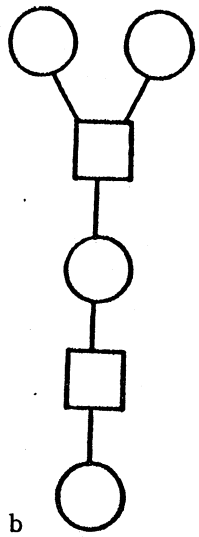
(')	OP	FORMAT 8 BITS	COMMENTAIRES	OP	FORMAT 16 BITS	COMMENTAIRES
1)	*	CLR m				
2)		LDAA n LDAB n LDAA m LDAB m			LDX n LDX m LDS n LDS m JMP n,X	
1)		STAA n STAB n STAA m STAB m			STX n STS n STX m STS m	
2)	* *	ADDA # n ADDB # n SUBA # n SUBB # n ANDA # n ANDB # n ORAA # n ORAB # n EORA # n EORB # n ABA # SBA # BITA # n BITB # n CMPA # n CMPB # n CBA ROLA ROLB RORA RORB			BRA n	
				*	CPX # m	

CLASSE 3

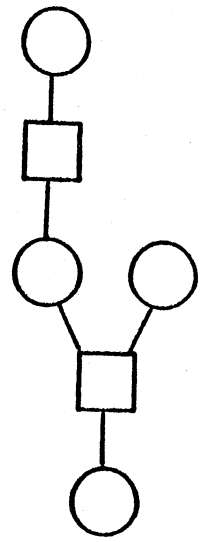
S-classe : a, b, c, d



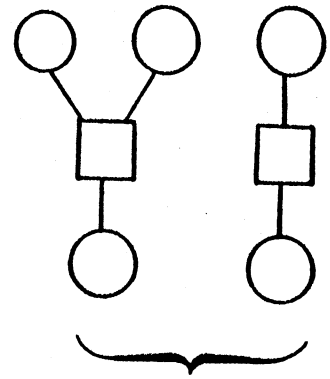
a



b



c

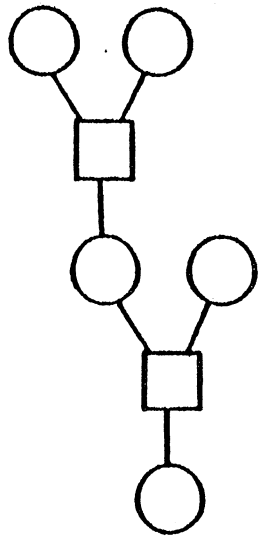


d

(T,T')	OP	FORMAT 8 BITS	COMMENTAIRES	OP	FORMAT 16 BITS	COMMENTAIRES
a) (3,2)	* * * *	ADCA # n ADCB # n SBCA # n SBCB # n				
b) (2,3)	*	TST m				
c) (2,2)		RTS RTI PULA PULB				
d) (2,2)		PSHA PSHB				

CLASSE 4

S-classe : a)

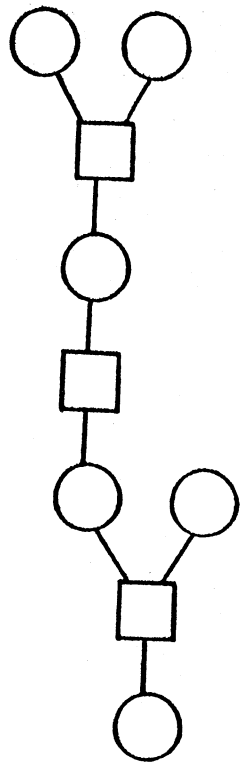


a

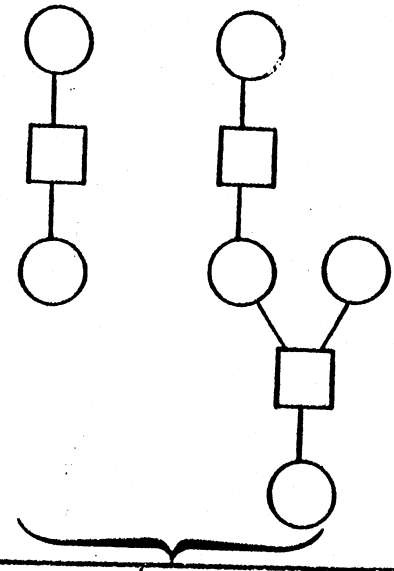
I')	OP	FORMAT 8 BITS	COMMENTAIRES	OP	FORMAT 16 BITS	COMMENTAIRES
1)	*	CLR n,X STAA n,X STAB n,X			STX n,X STS n,X	
2)		LDAA n,X LDAB n,X			LDX n,X LDS n,X	
	*	ADDA n			BCC n	
	*	ADDB n			BCS n	
	*	SUBA n			BEQ n	
	*	SUBB n			BGE n	
	*	ANDA n			BGT n	
	*	ORAA n			BHI n	
	*	ORAB n			BLE n	
	*	EORA n			BLS n	
	*	EORB n			BMI n	
	*	ANDB n			BNE n	
					BVC n	
					BVS n	
					BPL n	
					BLT n	
3)	*	BITA n				
	*	BITB n				
	*	BITA m				
	*	BITB m				
	*	CMPA n		*	CPX n	
	*	CMPB n		*	CPX m	
	*	CMPA m				
	*	CMPB m				

CLASSE 4

S-classe : b, c



b

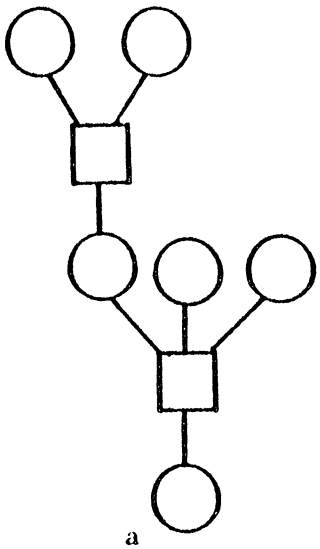


c

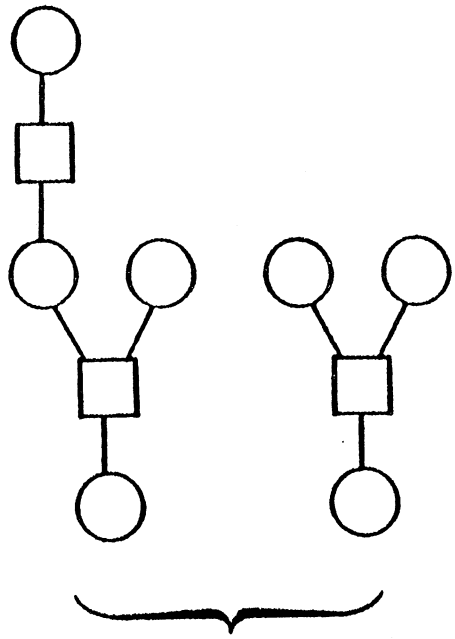
(T,T')	OP	FORMAT 8 BITS	COMMENTAIRES	OP	FORMAT 16 BITS	COMMENTAIRES
b) (1,1)	* * * * * * *	INC m DEC m NEG m ASL m ASR m LSR m COM m				
c) (2,2)					JSR m BSR m	

CLASSE 5

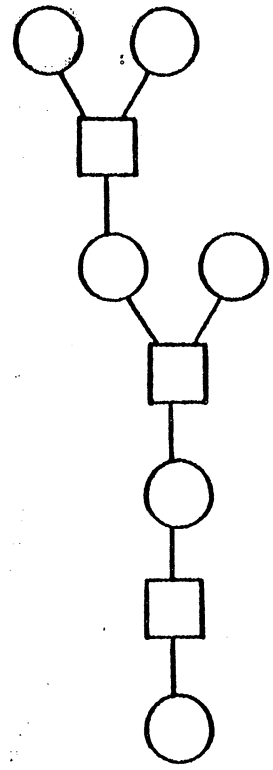
S-classe : a, b, c



a



b

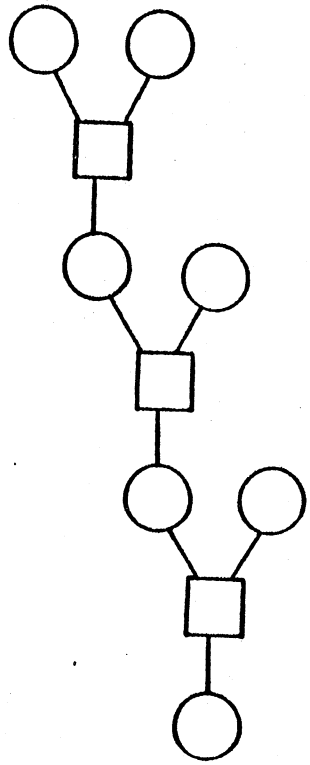


c

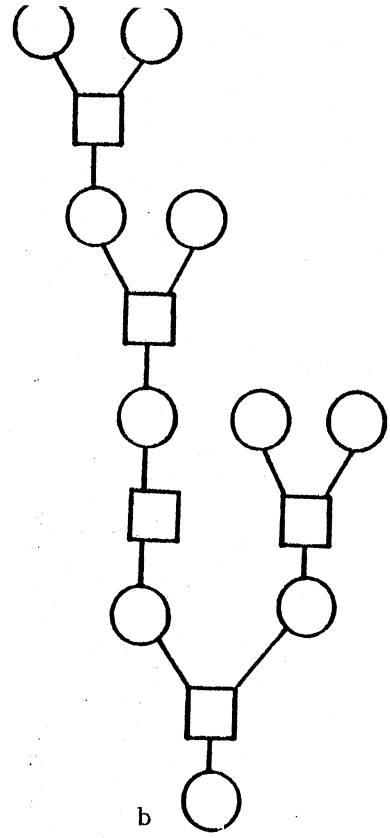
(3,2)	*	ADCA n				
	*	ADCB n				
	*	SBCA n				
	*	SBCB n				
	*	ADCA m				
	*	ADCB m				
	*	SBCA m				
	*	SBCB m				
(2,2)					JSR n,X	
(2,3)	*	TST n,X				

CLASSE 6

S-classe : a,b



a

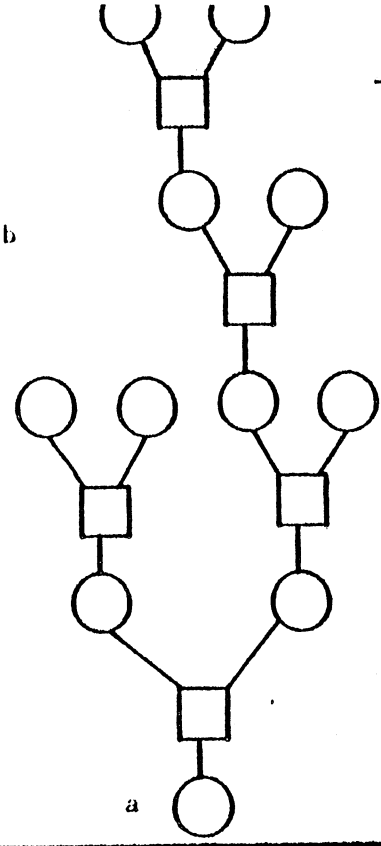


b

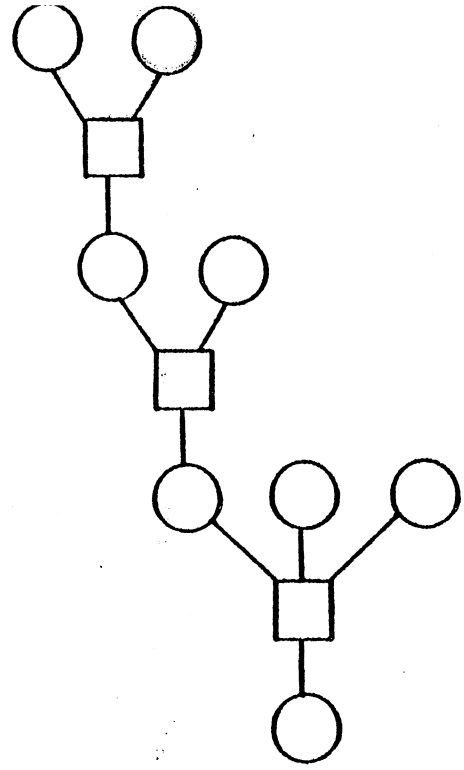
(T,T')	OP	FORMAT 8 BITS	COMMENTAIRES	OP	FORMAT 16 BITS	COMMENTAIRES
(2,2)	*	ADDA n,X				
	*	ADDB n,X				
	*	SBCA n,X				
	*	SBCB n,X				
	*	ANDA n,X				
	*	ANDB n,X				
	*	ORAA n,X				
	*	ORAB n,X				
	*	EORA n,X				
	*	EORB n,X				
(2,3)	*	BITA n,X				
	*	BITB n,X				
	*	CMPA n,X				
	*	CMPB n,X				
				*	CPX n,X	
(3,1)	*	ROR m				
	*	ROL m				
b)						
(2,1)	*	INC n,X				
	*	DEC n,X				
	*	NEG n,X				
	*	ASL n,X				
	*	ASR n,X				
	*	LSR n,X				
	*	COM n,X				

CLASSE 7

S-classe : a, b



a



b

T')	OP	FORMAT 8 BITS	COMMENTAIRES	OP	FORMAT 16 BITS	COMMENTAIRES
3,1)	*	ROR n,X				
	*	ROL n,X				
3,2)	*	ADCA n,X				
	*	ADCB n,X				
	*	SBCA n,X				
	*	SBCB n,X				

ANNEXE 5

EXTRAITS DU JEU D'INSTRUCTIONS DES MICROPROCESSEURS
MC6800 ET Z80

TABLE 3 – ACCUMULATOR AND MEMORY INSTRUCTIONS

OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A · M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B · M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A · M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B · M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A - M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B - M	•	•	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A - B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				M̄ → M	•	•	↕	↕	R	S
	COMA													43	2	1	Ā → A	•	•	↕	↕	R	S
	COMB													53	2	1	B̄ → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 - M → M	•	•	↕	↕	(1)	(2)
	NEGA													40	2	1	00 - A → A	•	•	↕	↕	(1)	(2)
	NEGB													50	2	1	00 - B → B	•	•	↕	↕	(1)	(2)
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add of BCD Characters into BCD Format	•	•	↕	↕	↕	(3)
Decrement	DEC							6A	7	2	7A	6	3				M - 1 → M	•	•	↕	↕	↕	•
	DECA													4A	2	1	A - 1 → A	•	•	↕	↕	↕	•
	DECB													5A	2	1	B - 1 → B	•	•	↕	↕	↕	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A⊕M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B⊕M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	↕	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	↕	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	↕	•
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA													36	4	1	A → M_{SP}, SP - 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → M_{SP}, SP - 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{SP} → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, M_{SP} → B	•	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3				M	•	•	↕	↕	↕	↕
	ROLA													49	2	1	A	•	•	↕	↕	↕	↕
	ROLB													59	2	1	B — C b7 — b0	•	•	↕	↕	↕	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	•	↕	↕	↕	↕
	RORA													46	2	1	A	•	•	↕	↕	↕	↕
	RORB													56	2	1	B — C b7 — b0	•	•	↕	↕	↕	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M	•	•	↕	↕	↕	↕
	ASLA													48	2	1	A	•	•	↕	↕	↕	↕
	ASLB													58	2	1	B — b7 b0 — 0 C	•	•	↕	↕	↕	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M	•	•	↕	↕	↕	↕
	ASRA													47	2	1	A	•	•	↕	↕	↕	↕
	ASRB													57	2	1	B — b7 b0 — C	•	•	↕	↕	↕	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M	•	•	R	↕	↕	↕
	LSRA													44	2	1	A	•	•	R	↕	↕	↕
	LSRB													54	2	1	B — 0 b7 b0 — C	•	•	R	↕	↕	↕
Store Acmltr	STAA				97	4	2	A7	6	2	B7	5	3				A → M	•	•	↕	↕	R	•
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A - M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B - M → B	•	•	↕	↕	↕	↕
Subtract Acmltrs	SBA													10	2	1	A - B → A	•	•	↕	↕	↕	↕
Subtr with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A - M - C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B - M - C → B	•	•	↕	↕	↕	↕
Transfer Acmltrs	TAB													16	2	1	A → B	•	•	↕	↕	R	•
	TBA													17	2	1	B → A	•	•	↕	↕	R	•
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M - 00	•	•	↕	↕	R	R
	TSTA													4D	2	1	A - 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B - 00	•	•	↕	↕	R	R
																		H	I	N	Z	V	C

ADDRESSING MODES

BOOLEAN/ARITHMETIC OPERATION

COND CODE REG

LEGEND

OP	Operation Code (Hexadecimal).	+	Boolean Inclusive OR
~	Number of MPU Cycles.	⊕	Boolean Exclusive OR
#	Number of Program Bytes.	M̄	Complement of M.
+	Arithmetic Plus.	→	Transfer Into
-	Arithmetic Minus.	0	Bit = Zero.
·	Boolean AND	00	Byte = Zero

M_{SP} Contents of memory location pointed to be Stack Pointer

Note - Accumulator addressing mode instructions are included in the column for IMPLIED addressing

CONDITION CODE SYMBOLS

H	Half-carry from bit 3.
I	Interrupt mask
N	Negative (sign bit)
Z	Zero (byte)
V	Overflow, 2's complement
C	Carry from bit 7
R	Reset Always
S	Set Always
↕	Test and set if true, cleared otherwise
•	Not Affected

TABLE 4 – INDEX REGISTER AND STACK MANIPULATION INSTRUCTIONS

POINTER OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BOOLEAN/ARITHMETIC OPERATION	5 H	4 I	3 N	2 Z	1 V	0 C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$X_H - M, X_L - (M + 1)$	•	•	⑦	↕	⑦	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	•	↕	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	•	↕	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M + 1) \rightarrow X_L$	•	•	⑨	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M + 1) \rightarrow SP_L$	•	•	⑨	↕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M + 1)$	•	•	⑨	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M + 1)$	•	•	⑨	↕	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

TABLE 5 – JUMP AND BRANCH INSTRUCTIONS

OPERATIONS	MNEMONIC	RELATIVE OP	RELATIVE ~	RELATIVE #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BRANCH TEST	5 H	4 I	3 N	2 Z	1 V	0 C
Branch Always	BRA	20	4	2										None	•	•	•	•	•	•
Branch If Carry Clear	BCC	24	4	2										C = 0	•	•	•	•	•	•
Branch If Carry Set	BCS	25	4	2										C = 1	•	•	•	•	•	•
Branch If = Zero	BEQ	27	4	2										Z = 1	•	•	•	•	•	•
Branch If ≥ Zero	BGE	2C	4	2										N ⊕ V = 0	•	•	•	•	•	•
Branch If > Zero	BGT	2E	4	2										Z + (N ⊕ V) = 0	•	•	•	•	•	•
Branch If Higher	BHI	22	4	2										C + Z = 0	•	•	•	•	•	•
Branch If ≤ Zero	BLE	2F	4	2										Z + (N ⊕ V) = 1	•	•	•	•	•	•
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	•	•	•	•	•	•
Branch If < Zero	BLT	2D	4	2										N ⊕ V = 1	•	•	•	•	•	•
Branch If Minus	BMI	2B	4	2										N = 1	•	•	•	•	•	•
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	•	•	•	•	•	•
Branch If Overflow Clear	BVC	28	4	2										V = 0	•	•	•	•	•	•
Branch If Overflow Set	BVS	29	4	2										V = 1	•	•	•	•	•	•
Branch If Plus	BPL	2A	4	2										N = 0	•	•	•	•	•	•
Branch To Subroutine	BSR	8D	8	2											•	•	•	•	•	•
Jump	JMP				6E	4	2	7E	3	3				} See Special Operations	•	•	•	•	•	•
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•
No Operation	NOP														•	•	•	•	•	•
Return From Interrupt	RTI										3B	2	1	Advances Prog Cntr. Only	•	•	•	•	•	•
Return From Subroutine	RTS										38	10	1				⑩			
Software Interrupt	SWI										3F	5	1	} See Special Operations	•	•	•	•	•	•
Wait for Interrupt	WAI										3E	9	1		•	⑪	•	•	•	•

TABLE 6 – CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

OPERATIONS	MNEMONIC	IMPLIED OP	IMPLIED ~	IMPLIED #	BOOLEAN OPERATION	5 H	4 I	3 N	2 Z	1 V	0 C
Clear Carry	CLC	0C	2	1	$0 \rightarrow C$	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	$0 \rightarrow I$	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	$0 \rightarrow V$	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	$1 \rightarrow C$	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	$1 \rightarrow I$	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	$1 \rightarrow V$	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	$A \rightarrow CCR$	⑫					
CCR → Acmltr A	TPA	07	2	1	$CCR \rightarrow A$	•	•	•	•	•	•

CONDITION CODE REGISTER NOTES:

(Bit set if test is true and cleared otherwise)

1 (Bit V) Test: Result = 10000000?
2 (Bit C) Test: Result = 00000000?
3 (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4 (Bit V) Test: Operand = 10000000 prior to execution?
5 (Bit V) Test: Operand = 01111111 prior to execution?
6 (Bit V) Test: Set equal to result of N⊕C after shift has occurred.
7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?
8 (Bit V) Test: 2's complement overflow from subtraction of MS bytes?
9 (Bit N) Test: Result less than zero? (Bit 15 = 1)
10 (All) Load Condition Code Register from Stack. (See Special Operations)
11 (Bit I) Set when interrupt occurs. If previously set, a Non Maskable Interrupt is required to exit the wait state.
12 (All) Set according to the contents of Accumulator A.

Microprocesseur Z80

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	OP-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T Cycles	Comments
LD r, r'	r ← r'	•	•	•	•	•	•	01 r r'	1	1	4	r, r' Reg.
LD r, n	r ← n	•	•	•	•	•	•	00 r 110 ← n →	2	2	7	000 B 001 C 010 D 011 E 100 H 101 L 111 A
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	01 r 110	1	2	7	
LD r, (IX+d)	r ← (IX+d)	•	•	•	•	•	•	11 011 101 01 r 110 ← d →	3	5	19	
LD r, (IY+d)	r ← (IY+d)	•	•	•	•	•	•	11 111 101 01 r 110 ← d →	3	5	19	
LD (HL), r	(HL) ← r	•	•	•	•	•	•	01 110 r	1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	•	•	•	•	11 011 101 01 110 r ← d →	3	5	19	
LD (IY+d), r	(IY+d) ← r	•	•	•	•	•	•	11 111 101 01 110 r ← d →	3	5	19	
LD (HL), n	(HL) ← n	•	•	•	•	•	•	00 110 110 ← n →	2	3	10	
LD (IX+d), n	(IX+d) ← n	•	•	•	•	•	•	11 011 101 00 110 110 ← d → ← n →	4	5	19	
LD (IY+d), n	(IY+d) ← n	•	•	•	•	•	•	11 111 101 00 110 110 ← d → ← n →	4	5	19	
LD A, (BC)	A ← (BC)	•	•	•	•	•	•	00 001 010	1	2	7	
LD A, (DE)	A ← (DE)	•	•	•	•	•	•	00 011 010	1	2	7	
LD A, (nn)	A ← (nn)	•	•	•	•	•	•	00 111 010 ← n → ← n →	3	4	13	
LD (BC), A	(BC) ← A	•	•	•	•	•	•	00 000 010	1	2	7	
LD (DE), A	(DE) ← A	•	•	•	•	•	•	00 010 010	1	2	7	
LD (nn), A	(nn) ← A	•	•	•	•	•	•	00 110 010 ← n → ← n →	3	4	13	
LD A, I	A ← I	•	‡	IFF	‡	0	0	11 101 101 01 010 111	2	2	9	
LD A, R	A ← R	•	‡	IFF	‡	0	0	11 101 101 01 011 111	2	2	9	
LD I, A	I ← A	•	•	•	•	•	•	11 101 101 01 000 111	2	2	9	
LD R, A	R ← A	•	•	•	•	•	•	11 101 101 01 001 111	2	2	9	

Instructions de transfert (8 bits)

Mnemonic	Symbolic Operation	Flags C	Z	M	S	P/V	N	H	Op-Code 76	543	210	No. of Bytes	No. of M Cycles	No. of T States
LD dd, nn	dd ← nn	•	•	•	•	•	•	•	00	dd0	001	3	3	10
									—	n	—			
									—	n	—			
LD IX, nn	IX ← nn	•	•	•	•	•	•	•	11	011	101	4	4	14
									00	100	001			
									—	n	—			
									—	n	—			
LD IY, nn	IY ← nn	•	•	•	•	•	•	•	11	111	101	4	4	14
									00	100	001			
									—	n	—			
									—	n	—			
LD HL, (nn)	H ← (nn+1)	•	•	•	•	•	•	•	00	101	010	3	5	16
	L ← (nn)								—	n	—			
									—	n	—			
LD dd, (nn)	dd_H ← (nn+1)	•	•	•	•	•	•	•	11	101	101	4	6	20
	dd_L ← (nn)								01	dd1	011			
									—	n	—			
									—	n	—			
LD IX, (nn)	IX_H ← (nn+1)	•	•	•	•	•	•	•	11	011	101	4	6	20
	IX_L ← (nn)								00	101	010			
									—	n	—			
									—	n	—			
LD IY, (nn)	IY_H ← (nn+1)	•	•	•	•	•	•	•	00	101	010	4	6	20
	IY_L ← (nn)								00	101	010			
									—	n	—			
									—	n	—			
LD (nn), HL	(nn+1) ← H	•	•	•	•	•	•	•	00	100	010	3	5	16
	(nn) ← L								—	n	—			
									—	n	—			

Mnemonic	Symbolic Operation	Flags C	Z	M	S	P/V	N	H	Op-Code 76	543	210	No. of Bytes	No. of M Cycles	No. of T States	Comments	
LD (nn), dd	(nn+1) ← dd_H	•	•	•	•	•	•	•	11	101	101	4	6	20		
	(nn) ← dd_L								01	dd0	011				dd	Pair
									—	n	—				00	BC
									—	n	—				01	DE
LD (nn), IX	(nn+1) ← IX_H	•	•	•	•	•	•	•	11	011	101	4	6	20	10	HL
	(nn) ← IX_L								00	100	010				11	SP
									—	n	—					
									—	n	—					
LD (nn), IY	(nn+1) ← IY_H	•	•	•	•	•	•	•	11	111	101	4	6	20		
	(nn) ← IY_L								00	100	010					
									—	n	—					
									—	n	—					
LD SP, HL	SP ← HL	•	•	•	•	•	•	•	11	111	001	1	1	6		
LD SP, IX	SP ← IX	•	•	•	•	•	•	•	11	011	101	2	2	10		
									11	111	001					
LD SP, IY	SP ← IY	•	•	•	•	•	•	•	11	111	101	2	2	10		
									11	111	001					
PUSH qq	(SP-2) ← qq_L	•	•	•	•	•	•	•	11	qq0	101	1	3	11	qq	Pair
	(SP-1) ← qq_H														00	BC
PUSH IX	(SP-2) ← IX_L	•	•	•	•	•	•	•	11	011	101	2	4	15	01	DE
	(SP-1) ← IX_H								11	100	101				10	HL
PUSH IY	(SP-2) ← IY_L	•	•	•	•	•	•	•	11	111	101	2	4	15	11	AF
	(SP-1) ← IY_H								11	100	101					
POP qq	qq_H ← (SP+1)	•	•	•	•	•	•	•	11	qq0	001	1	3	10		
	qq_L ← (SP)															
POP IX	IX_H ← (SP+1)	•	•	•	•	•	•	•	11	011	101	2	4	14		
	IX_L ← (SP)								11	100	001					
POP IY	IY_H ← (SP+1)	•	•	•	•	•	•	•	11	111	101	2	4	14		
	IY_L ← (SP)								11	100	001					

Instructions de transfert (16 bits)

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	1	1	4	
EXX	$\begin{pmatrix}BC\\DE\\HL\end{pmatrix}\leftrightarrow\begin{pmatrix}BC'\\DE'\\HL'\end{pmatrix}$	•	•	•	•	•	•	11 011 001	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔ (SP+1) L ↔ (SP)	•	•	•	•	•	•	11 100 011	1	5	19	
EX (SP), IX	$IX_H \leftrightarrow (SP+1)$ $IX_L \leftrightarrow (SP)$	•	•	•	•	•	•	11 011 101 11 100 011	2	6	23	
EX (SP), IY	$IY_H \leftrightarrow (SP+1)$ $IY_L \leftrightarrow (SP)$	•	•	•	•	•	•	11 111 101 11 100 011	2	6	23	
LDI	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	①	•	0	0	11 101 101 10 100 000	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	0 ①	•	0	0	11 101 101 10 110 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	①	•	0	0	11 101 101 10 101 000	2	4	16	
LDDR	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	0 ①	•	0	0	11 101 101 10 111 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A - (HL) HL ← HL+1 BC ← BC-1	•	②	①	②	①	②	11 101 101 10 100 001	2	4	16	
CPIR	A - (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	②	①	②	①	②	11 101 101 10 110 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A - (HL) HL ← HL-1 BC ← BC-1	•	②	①	②	①	②	11 101 101 10 101 001	2	4	16	
CPDR	A - (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	②	①	②	①	②	11 101 101 10 111 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Instructions de transfert de blocs et d'échange des bancs des registres.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU les rapports de présentation de

. Madame G. SAUCIER, Professeur

. Monsieur GRILLOT, Ingénieur à E.M.D

Monsieur Raul VELAZCO

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, Spécialité "Génie Informatique".

Fait à Grenoble, le 2 mars 1982

Le Président de l'I.N.P.-G.

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble