Sc N 71/34 A

UNIVERSITÉ DE NANCY FACULTÉ DES SCIENCES

SIMULATION DE MACHINES POUR L'ENSEIGNEMENT DE LA PROGRAMMATION

-=-=-=-=-

THÈSE

UNIVERSITE NANCY BIBLIOTHÈQUE

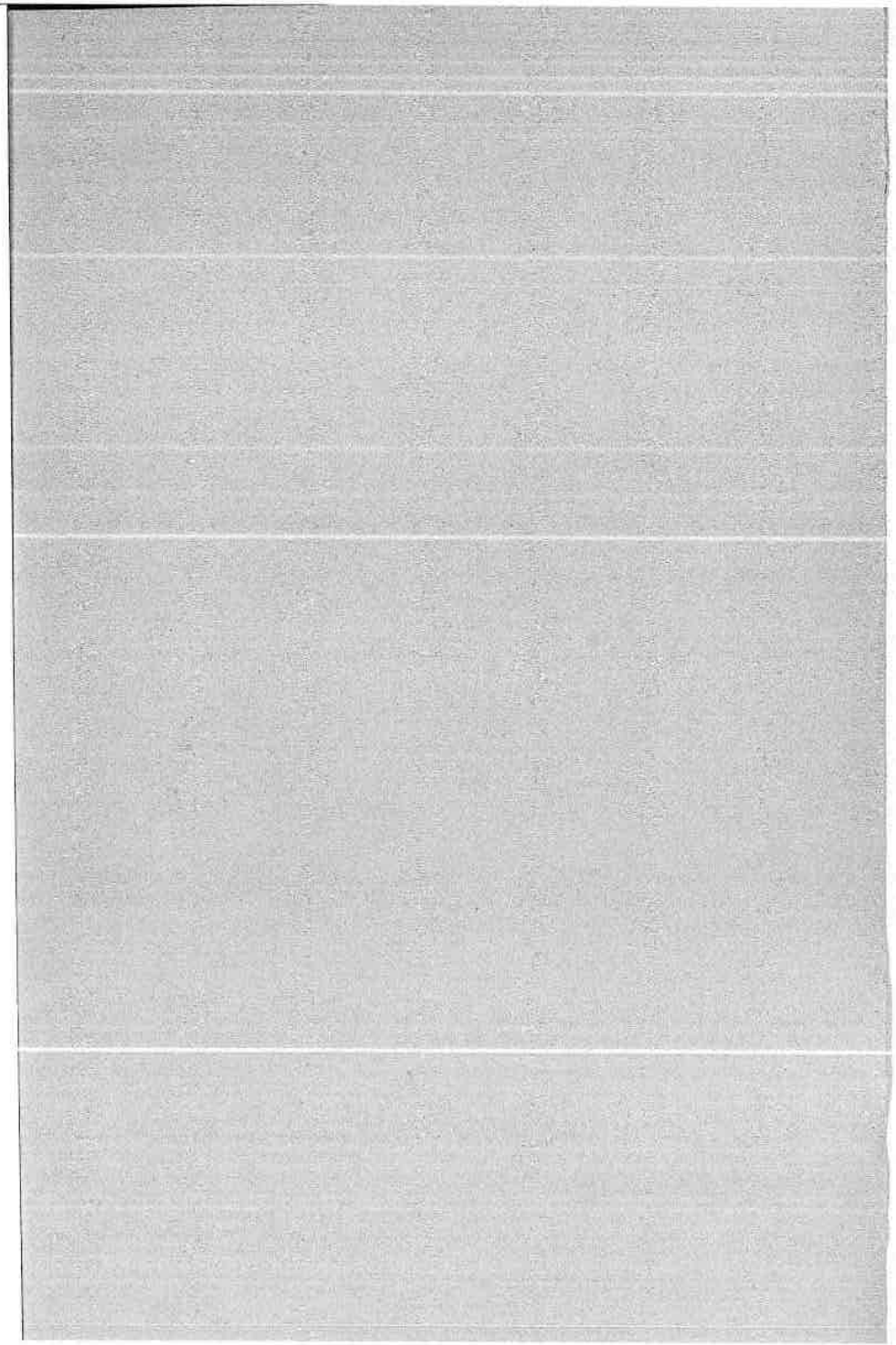
pour l'obtention du
DOCTORAT de SPECIALITE MATHEMATIQUES (3ème CYCLE)

Soutenue devant le Jury le 26 Juin 1971

par

Madame DENDIEN Marie-Claude

Jury : Mr LEGRAS Président
Mr PAIR Examinateurs
Mr DERNIAME



UNIVERSITE DE NANCY FACULTE DES SCIENCES

SIMULATION DE MACHINES POUR L'ENSEIGNEMENT DE LA PROGRAMMATION

UNIVERSITE NANCY BIBLIOTHEQUE

par Madame DENDIEN Marie-Claude

Que Monsieur le Professeur LEGRAS, Directeur de l'Institut Universitaire de Calcul Automatique, trouve ici l'expression de ma profonde gratitude pour la bienveillance qu'il m'a toujours témoignée et l'intérêt qu'il a porté à ce travail.

Je remercie vivement Monsieur PAIR, pour les conseils qu'il a bien voulu me donner et l'honneur qu'il me fait en participant au jury.

Ce travail a été effectué sous la direction de Monsieur DERNIAME. Qu'il trouve ici tous mes remerciements pour l'aide et les conseils amicaux qu'il m'a prodigués et la formation qu'il m'a donnée.

Les Travaux Pratiques sur ILM70 on été le fruit d'un travail commun avec Monsieur J. DUCLOY. Je le remercie pour ses suggestions et sa sympathique collaboration.

Je remercie également Madame KUGLER qui a assuré avec rapidité et soin la réalisation matérielle de cette thèse.

SOMMAIRE

Chapitre I - PROBLEMES POSES PAR L'ENSEIGNEMENT de la PROGRAMMATION EN LANGAGE MACHINE

Introduction

I. 1 Idées fondamentales à faire acquérir.

I. 2 Progression dans l'acquisition de ces notions.

I. 3 Utilisation de machines simulées.

Chapitre II - MACHINE M. I. P.

II. 1 Présentation

II. 2 Description du programme de simulation.

II. 3 Modifications possibles du programme de simulation.

ANNEXE 1 : Code et description des instructions de M. I. P.

Chapitre III - DESCRIPTION DES MACHINES SIMULEES

Introduction - Présentation

III. 1 ILM70-1 - Description.

III. 2 ILM70-1 - Fonctionnement.

III. 3 ILM70-2 - Description.

III. 4 ILM70-3 - Description.

ANNEXE 2 : Instructions du CII 10070 : décalage, déplacement de chaines d'octets, comparaison de chaines d'octets.

Chapitre IV - UTILISATION D'ILM70

IV. 1 Initiation à la programmation : utilisation d'ILM70-1.

IV. 2 Apprentissage des techniques de programmation : utilisation d'ILM70-2.

IV. 3 Utilisation d'ILM70-3 : Problèmes de chargement, d'assemblage, d'édition de liens.

Chapitre V - Simulation d'ILM70

PLAN

INTRODUCTION

V. 1 NIV1 et NIV2 - organisation générale.

V. 1. 1 Chargement (CHARG)

V. 1. 2 Initialisations (INIT)

V. 1. 3 Interprête (INTER)

V. 1. 3. 1 Représentation des éléments d'ILM70.
V. 1. 2. 2 Description de l'interprète :

CINST
PROG
TAI
AIG a) Représentation du jeu d'instructions
b) Description de l'aiguillage.

SEQ
BRCHT a) dans NIV1
b) dans NIV2

TBAL a) dans NIV1
b) dans NIV2.

V. 1. 4 Quelques remarques sur l'écriture de NIV1 et NIV2.

V. 2 NIV3.

V. 2. 1 Instructions sur chaines d'octets.

V. 2. 2 Simulation des E/S en simultanéité.

a) Principe
b) Rôle des indicateurs de temps d'attente
c) Simulation de SIØ et TIØ
d) Gestion des E/S en attente
e) Exemple.

CHAPITRE I

PROBLEMES POSES PAR L'ENSEIGNEMENT DE LA PROGRAMMATION EN LANGAGE MACHINE

Le contenu et l'organisation d'un enseignement de la programmation en langage machine puis symbolique sont fonctions de l'auditoire auquel on s'adresse, du niveau en informatique que l'on veut atteindre et du matériel disponible Un tel enseignement est rarement une fin en soi, et il doit s'insérer généralement dans un cadre plus vaste d'enseignement de l'informatique. Dans cette hypothèse, il faut répondre aux questions suivantes :

l'enseignement de la programmation en langage machine est-il nécessaire ? dans l'affirmative, à quel degré faut-il le développer ? à quel moment doit-on l'introduire ?

Il semble que l'étude de la programmation en langage machine soit indispensable si on veut donner une idée juste et complète du fonctionnement d'un ordinateur : Par la suite les connaissances ainsi obtenues permettent une meilleure utilisation des langages évolués. Un troisième intérêt est de faire découvrir à l'auditoire la nécessité, le rôle et le contenu d'un software de base, c'est-à-dire des chargeurs, des assembleurs, des sous-programmes de service etc... L'utilisation effective d'un tel software permet de ressentir le besoin d'un software plus évolué et d'en définir le rôle. Ceci permet d'introduire par exemple les notions d'édition de liens ou de système d'enchainement de travaux.

Ces trois points de vue correspondent à peu près à trois types d'enseignement de l'informatique : l'initiation superficielle, dans un but culturel, la formation d'utilisateurs devant programmer en langage évolué, l'enseignement destiné à de futurs spécialistes. L'importance accordée à cette étude peut aller de la simple présentation du déroulement d'un programme en langage machine, dans le premier cas, à l'étude approfondie et à la pratique effective dans le dernier cas. Quelque soit le développement qui lui est donné, il n'est pas souhaitable de débuter l'étude de l'informatique par la programmation en langage machine. C'est en effet un sujet aride, qui demande des analyses trés détaillées. La programmation en langage évolué simple (du type Fortran ou Algol) a l'avantage de donner à l'auditoire une idée plus juste de la programmation habituelle (un utilisateur courant s'exprime peu en langage machine) tout en le familiarisant avec l'analyse d'un problème en vue de sa programmation ainsi qu'avec les notions essentielles telles que les branchements, les boucles etc...

I.1 Idées fondamentales à faire acquérir

Nous allons faire ici l'inventaire des élèments pouvant intervenir dans un cours de programmation au niveau du langage machine. N'ont été envisagées que les notions de base, permettant d'aborder sans problème la programmation en n'importe quel langage machine ou symbolique :

- Une description des différents éléments d'un ordinateur :
  mémoire centrale, unité centrale, périphérique, mot mémoire, adresse, registre.
- Une description du fonctionnement d'un ordinateur :
  instruction, rôle du (ou des) registre (s) instruction (s), et du compteur ordinal, enchainement des instructions, branchements.
- Un apprentissage des techniques de bases de la programmation :
  Boucles, compteurs, utilisation de l'adressage indirect et de l'indexation, sous-programmes fermés et probléme du passage de leurs paramètres.
- Une étude des échanges :
  caractères et octets, manipulation d'octets et de chaines d'octets instructions d'entrée -sortie , interruptions, simultanéités, programmation d'une unité d'échange.
- Un exposé du problème posé par l'initialisation d'un ordinateur :
  différentes méthodes d'initialisation. Notion de chargeur câblé, de chargeur programmé. Translation d'un programme.
- Une étude de la correspondance langage symbolique - langage machine
  assembleurs, directives d'assemblage, macro-instructions et sous-programmes ouverts.

Pour chaque auditoire, il faut choisir parmi ces notions, et déterminer dans quel ordre elles doivent être exposées.

I.2 Progression dans l'acquisition des ces notions :

La progression présentée ici est celle suivie dans le certificat d'informatique générale, C2 de la maitrise d'Informatique à Nancy.

Le programmeur débutant en langage machine doit assimiler quasi simultanément des concepts nouveaux relativement simples, mais déroutants. Il s'agit des éléments permettant de comprendre le déroulement d'un programme en langage machine. A cela s'ajoute l'utilisation du système de numération binaire et de sa représentation héxadécimale ou octale.

Comme souvent en informatique, mais cela est encore plus sensible pendant cette initiation, certaines de ces notions sont fortement imbriquées et il est parfois difficile de définir un enchainement logique dans leur présentation.

Par exemple, pour bien comprendre le fonctionnement d'une machine, même très simple, il faut avoir écrit quelques programmes ; mais pour écrire ces programmes il faut avoir compris ce fonctionnement ...

On voit donc apparaître deux modes de progression dans l'enseignement du langage machine :

- une progression logique dans l'acquisition de nouvelles connaissances (linéaire)
- une progression "en plusieurs passages" : certaines notions devront être abordées superficiellement d'abord pour permettre d'acquérir des connaissances nécessaires pour les traiter complétement plus tard. On dépose ainsi plusieurs couches successives de savoir qui vont en se précisant. Il est essentiel que dés le début de l'enseignement la cohérence du fonctionnement d'un ordinateur et son caractère parfaitement déterministe soient mis en évidence On doit également préciser le rôle de chaque composant, et en particulier des périphériques. Ici apparait un problème qui se pose souvent lors d'une initiation à la programmation, que les langages étudiés soient évolués ou non. L'utilisation des entrées-sorties demande un volume de connaissances important. Elle est pourtant souhaitable le plus tôt possible si on veut éviter une confusion entre les constantes, qui figent le traitement et les données qui font l'objet du traitement. Une solution consiste à définir une instruction de lecture ou d'écriture simple qui correspondra au de l'organigramme :

elle présente l'inconvénient de masquer les problèmes de programmation des entrées-sorties. On peut également donner à l'auditoire une séquence d'instructions à insérer dans les programmes en précisant qu'on étudiera ces instructions plus tard, ou, ce qui est plus commode, mettre ces instructions sous forme d'un sous-programme sans paramètre. L'expérience prouve que l'utilisation de sous-programmes sans paramètre ne pose aucun problème à l'auditoire dès l'instant où celui-ci a assimilé la notion de branchement. Cette dernière solution à l'avantage de ne pas donner une illusion de facilité, tout en étant d'un emploi aisé.

Le jeu d'instructions utilisé dans les premiers programmes sera volontairement réduit, mais devra permettre l'utilisation de compteurs et la programmation de boucles. L'écriture de quelques programmes de ce type, très simples mais demandant de calculer des adresses et de modifier des instructions conduit naturellement à penser à des dispositifs automatiques pour faire ces modifications. On peut alors parler d'adressage indirect (remplacement d'une adresse par le contenu du mot qu'elle désigne) et d'indexation (addition ou soustraction d'une quantité à une adresse).

L'auditoire possède alors tous les éléments pour écrire des sous-programmes et étudier les différentes manières de conserver l'adresse de retour et d'accéder aux paramètres. Il vaut mieux placer cette étude dès maintenant car cette notion a déjà été abordée superficiellement et il est souhaitable de la préciser le plus tôt possible ; elle fournit l'occasion d'excellents exercices sur l'adressage indirect ; elle permet l'écriture de programmes importants en travaux pratiques et fournit un lien commode avec une analyse et une description modulaire.

Ces connaissances acquises, l'introduction de notions nouvelles pose moins de problèmes.

Il reste à parler des échanges de l'ordinateur avec l'extérieur, et de la correspondance Langage symbolique - Langage machine. Il est intéressant de passer tôt au langage symbolique, plus agréable et d'utilisation plus normale que celle du langage machine. En particulier, l'étude des entrées-sorties peut trés bien se faire en utilisant un langage symbolique.

Le langage symbolique peut d'ailleurs être présenté, dés le début de l'enseignement, comme un intermédiaire commode entre l'organigramme et le programme en langage machine ; dés que l'auditoire dispose de connaissances suffisantes, on lui fait écrire un assembleur simple. A partir de ce niveau, il n'utilisera plus qu'un (ou des) langage symbolique. Pour écrire un assembleur, il faut savoir traiter une chaine de caractères, et surtout savoir lire les 80 caractères d'une carte. On est donc amené à parler d'octets, de chaines d'octets, et un peu des entrées-sorties de caractères.

Dans le cas d'auditoires ne recherchant pas une spécialisation poussée, on pourra se limiter à la description du fonctionnement d'une machine et du rôle d'un assembleur. Cependant, s'il s'agit de futurs utilisateurs de langages évolués, il est souhaitable de poursuivre l'enseignement jusqu'à l'étude des sous-programmes et des traitements de chaines de caractères.

I, 3 Utilisation de machines simulées

En général les machines dont on dispose sont trop complexes pour être présentées à un débutant. Il est donc naturel de penser à définir une machine très simple mais complète, dont le langage machine, aussi dépouillé que possible, permette d'écrire des programmes et d'assimiler les premières notions importantes. Sur ce type de machines, certaines fonctions (par exemple les entrées-sorties) seront volontairement laissées dans l'ombre. Elles seront néanmoins présentes, et on y reviendra par la suite. On passera ensuite à la programmation de machines plus complexes et plus perfectionnées pour aboutir à l'utilisation des machines actuelles.

La définition d'une telle suite de machines est un problème délicat : il est en effet souhaitable de donner à la fois une expérience pratique et un enseignement général à l'apprenti programmeur. Ces deux impératifs sont difficilement conciliables dans l'enseignement du langage machine, car expérience pratique signifie programmation d'une machine particulière, donc utilisation d'un langage machine particulier... La plupart des langages machine ont des caractéristiques communes, et les notions de base, proches de la technologie sont constantes : par exemple, les notions de mot-mémoire, d'adresse, de registre, etc... Au niveau élémentaire, il sera donc relativement simple de définir un langage machine où l'on ne trouvera que des caractéristiques communes à tous les langages machine . Mais au fur et à mesure que l'on progressera dans l'enseignement, on sera amené à faire un certain nombre de choix dans la définition des langages machine utilisés en travaux pratiques, le cours gardant une optique plus générale. Ces choix sont évidemment fonction du matériel qui sera utilisé par la suite. Il s'agit, à Nancy, d'un CII 10070.

Il est essentiel que le passage d'une machine à une autre se fasse sans perte de temps inutile, en particulier qu'une instruction donnée, à partir du moment où l'on commence à l'utiliser, ait une écriture unique dans les diverses machines. On est donc amené à définir une suite de sous-ensembles croissants dans le langage machine de l'ordinateur qui sera utilisé plus tard. Ces sous-ensembles doivent être cohérents, comme pour une machine réelle et leur utilisation ne doit pas donner de mauvaises habitudes de programmation. En particulier, il faudra éviter, dans les premiers sous-ensembles, des simplifications pouvant être source d'erreurs ultérieures. Par exemple, dans

ILM70-1 présentée au chapitre III, l'instruction de multiplication peut sembler inutilement compliquée, mais il est préférable de l'introduire sous sa forme définitive plutôt que de la changer en cours de progression.

Dans le cas où l'auditoire veut se limiter à une initiation, l'utilisation du plus petit de ces sous-ensembles n'est donc pas souhaitable. On pourra alors définir une machine spécifique, extrémement simplifiée, on pourra même utiliser une machine décimale, pour éviter les difficultés que crée l'utilisation du système de numération binaire. Un exemple d'une telle machine est donné au chapitre suivant. On trouvera au chapitre III la description d'une suite de 3 machines fictives ILM70-1, ILM70-2, ILM70-3 définies au contraire selon les principes de progression précédemments exposés.

CHAPITRE II

MACHINE M I P

La machine M I P (Machine d'Initiation à la Programmation) est une machine décimale destinée à familiariser l'étudiant avec les notions de mémoire, d'instruction, de séquence d'instructions représentant un algorithme, de tests, de boucles.

Le programme de simulation de cette machine est écrit en Fortran. Il peut être utilisé sur tout ordinateur permettant l'emploi en Fortran, de réels ayant au moins 8 chiffres significatifs.

II.1 Présentation

II.1.1 Description :

C'est une machine décimale à une adresse.

Sa mémoire centrale est formée de 1000 mots de 9 digits dont les adresses varient de 0 à 999. Les nombres sont représentés sous la forme signe et valeur absolue de 8 chiffres. Le signe est contenu dans le digit le plus à gauche (plus fort poids).

L'unité centrale est composée de :

- 1 registre où se trouve l'adresse de l'instruction en cours, et qu'on appellera compteur ordinal.
- 1 ensemble de 4 registres où sont réparties les différentes zones de l'instruction en cours.
- 1 accumulateur (1 signe et 8 chiffres) complété par une extension.
- 9 registres d'index.
- 1 jeu de 18 instructions ; (voir Annexe 1). On peut répartir ces instructions en :

instructions de transfert :

chargement de l'accumulateur
chargement de l'extension
rangement de l'accumulateur
rangement de l'extension.

instructions arithmétiques :

addition
soustraction
multiplication
division

instructions de branchement :

branchement inconditionnel

2 branchements sur condition

instructions de manipulation de registres d'index :

Mise à zéro

chargement

rangement

modification

instructions de lecture et d'écriture :

lecture d'un mot sur carte

écriture d'un mot sur machine à écrire

instruction d'arrêt.

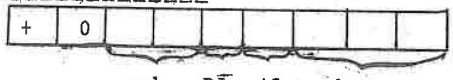
- Ses périphériques sont :

un lecteur de cartes en entrée

un clavier de machine à écrire en sortie.

II. 1. 2 Programmation

II. 1. 2. 1 Format d'une instruction



La zone code (3e et 4e digit) contient le code de l'instruction. (Voir annexe 1)
La zone RI, registre d'index, (5e digit) contient un chiffre décimal de 0 à 9. Si ce chiffre est différent de zéro, il désigne l'un des 9 registres d'index.
La zone AI, adressage indirect, (6e digit) contient un 0 ou un 1. Si ce nombre est 1, il y a adressage indirect.
La zone adresse (3 derniers digits) contient une adresse, c'est-à-dire un nombre décimal compris entre 0 et 999.

II. 1. 2. 2 Adressage :

L'utilisation de l'indexation est impossible dans le cas d'instructions de manipulation de registre d'index. La zone RI de ces instructions (5e digit) indique quel est le registre d'index concerné.

Lorsqu'il y a indexation et indirection dans une même instruction, l'adressage indirect précède l'indexation.

II. 1. 2. 3 Présentation d'un programme

Un programme est perforé sur cartes, à partir de la première colonne, à raison d'une instruction par carte.

Les instructions sont implantées en mémoire à partir de l'adresse 0.

La dernière instruction exécutée doit être l'instruction d'arrêt 07.
La fin d'un programme est repérée par une carte de commande : une carte de commande est formée de 1 chiffre suivi de 8 zéros. On peut avoir

000000000 : cette carte déclenche l'exécution du programme
100000000 : listing du programme puis exécution
200000000 : exécution du programme avec analyse
300000000 : listing puis exécution avec analyse.

II. 1. 2. 4 Analyse et listing

Une demande de listing provoque l'impression de toutes les cartes qui ont été lues avant la carte de commande : il y a donc impression du programme.

Une demande d'analyse provoque durant toute l'exécution du programme :

1) pour toutes les instructions, l'impression, au moment de l'exécution, du code et de l'adresse effective.

2) pour les instructions de manipulation de registres d'index, l'impression du numéro du registre d'index concerné.

3) pour les instructions modifiant le contenu d'un registre, (accumulateur, extension, index) l'impression du nouveau contenu.

II. 1. 2. 5 - Diagnostics d'erreur :

Pour faciliter la recherche des erreurs, leur détection donne lieu à un bref commentaire. Notons que nous ne cherchons que des erreurs qui sont normalement détectées par câblage (déroutements).

CO INCONNU le code de l'instruction en cours ne correspond pas à une des 18 instructions de la machine B.

RI INEXISTANT dans une instruction de manipulation de registre d'index on a trouvé un 0 dans la zone RX.

AI INCORRECT la zone AI de l'instruction en cours est différente de 0 ou 1.

ADRESSE NEGATIVE
DEBORDEMENT MEMOIRE } L'adresse effective de l'instruction en cours n'est pas comprise entre 0 et 999.
DIVISION PAR ZERO : division entière incorrecte.
ENTIER INCORRECT : au cours d'une opération de lecture on a rencontré dans les 9 premières colonnes de la carte un caractère autre qu'un signe ou un chiffre, ou un signe ailleurs qu'en première colonne.

II. 1. 2, 6 - Interruption d'un programme en cours

Il est nécessaire de pouvoir interrompre un programme qui boucle anormalement sans être obligé d'interrompre le programme de simulation. Deux options ont été programmées qui correspondent à 2 types de machines réelles utilisées pour la simulation. Pour la simulation sur CAE 510, où un système de clés est accessible à l'opérateur, une clé est testée chaque fois que l'on change d'instruction. Si elle est positionnée à 1, le programme en cours est interrompu ; lorsqu'on la remet à zéro on commence la lecture du programme suivant.

Pour la simulation sur CII 10070, il a fallu mettre en paramètre du programme de simulation le nombre maximum d'instructions qui pouvaient être exécutées par un programme. Lorsque ce nombre est atteint le programme est abandonné.

II. 2 Description du programme de simulation

Le programme de simulation de la machine MIP a été écrit en Fortran IV et utilisé sur CAE 510 et CII 10070. Sur CAE 510, une première version de ce programme chargeait et exécutait les programmes en langage machine MIP qui lui étaient présentés jusqu'à ce qu'on l'arrête manuellement. Ce programme a été modifié pour l'utilisation sur CII 10070 : la deuxième version charge et exécute 1 programme ; le programme de simulation est stocké sur disque sous forme de module de chargement et on le charge chaque fois qu'on veut passer un programme en langage machine MIP.

II. 2. 1 Organisation générale

Les traitements successifs à effectuer sont les suivants :

a) Initialisations propres au programme de simulation (par exemple, lecture du nombre maximum d'instructions machine à exécuter).

b) Initialisations propres au programme qui va être chargé et exécuté (compteur ordinal, etc...)

c) Lecture et implantation du programme en langage machine (1).

d) Exécution en mode impératif du programme en langage machine jusqu'à la rencontre d'une instruction d'arrêt.

e) (dans la première version) retour au traitement b).

Avant de détailler ces traitements, il est nécessaire de préciser la représentation dans le programme de simulation des principaux éléments de la machine MIP.

II. 2. 2 Représentation dans le programme de simulation des principaux éléments de la machine

- La mémoire centrale est représentée par le tableau d'entiers : MEM de dimension 2000.

En effet sur CAE 510, les entiers sont compris entre - 131062 et + 131061. Il a fallu représenter un mot de la machine MIP sur deux entiers dans le programme de simulation : un entier pour les 4 poids forts du mot et un entier pour les 4 poids faibles. Le contenu du mot MIP d'adresse n sera rangé dans MEM (2n+1) pour les poids forts, MEM (2n+2) pour les poids faibles, ces deux entiers étant obligatoirement de même signe.

- l'Unité centrale : le compteur ordinal est représenté par l'entier ICØR. Cet entier a pour valeur l'indice de l'élément de MEM où se trouvent les forts poids de l'instruction en cours. Il progresse de 2 lorsqu'on passe à l'instruction suivante.

Avant d'être exécutée, l'instruction dont l'adresse est dans le compteur ordinal est éclatée dans 4 registres qui sont représentés par 4 entiers : ICØP où on trouve le contenu de la zone "code", IRI où se trouve le contenu de la zone "RI", IAI où se trouve le contenu de la zone "AI", IAD où se trouve le contenu de la zone "adresse". Elle constitue alors l'instruction en cours.

L'accumulateur est représenté par 2 entiers, IAC1 pour les poids forts, IAC2 pour les poids faibles.

L'extension est représenté par IEX1 pour les poids forts, IEX2 pour les poids faibles.

Les 9 registres d'index sont représentés par 2 tableaux d'entiers de dimension 9, IND1 pour les poids forts, IND2 pour les poids faibles.

Le programme utilise par ailleurs une variable logique AN qui indique si l'analyse est demandée ou non, et une variable entière IEF qui contient l'indice du tableau MEM correspondant aux poids forts de l'opérande effectif, c'est-à-dire :

1 + 2 x adresse effective de l'instruction en cours.

II. 2. 3 Détail des différents traitements :

a) Initialisations propres au programme de simulation

- Lecture d'un tableau IC contenant les codes des 12 caractères utilisés en entrée-sortie : 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, +, -. Il occupe les 12 premiers entiers du common.

- Lecture d'un entier NMI qui est le nombre maximum d'instructions qui peuvent être exécutées au cours d'un programme. (utilisé dans le cas de la simulation sur CII 10070).

b) Initialisations propres au programme à charger :

- AN = . FALSE .
- ICØR = + 1
- NCI = 0 NCI comptera le nombre d'instructions exécutées au cours du programme, dans le cas de la simulation sur CII 10070.

c) Lecture et implantation du programme :

Les cartes sont lues selon le format 9A1. Leur contenu est converti par le sous-programme CØNV :

CØNV transforme un nombre décimal représenté par 9 caractères (rangés dans le tableau IL) en ce même nombre représenté par 2 entiers : 4 poids forts dans les premiers, 4 poids faibles dans le second ; ces deux entiers seront rangés dans Mem (icor) et Mem (icor + 1). On fait progresser icor de 2. Si lors de la conversion on rencontre un caractère qui ne peut appartenir à un nombre décimal, il y a impression du diagnostic d'erreur ENTIER INCORRECT et le programme est abandonné.

d) Exécution en mode interprétatif du programme en langage machine :

d. 1 initialisation du compteur ordinal : la lecture d'une carte de commande (dont le premier caractère est 0, 1, 2 ou 3) provoque éventuellement l'écriture des (icor-1)/2 mots lus (listing du programme) et (ou) l'affectation à AN de la valeur . TRUE., puis systématiquement la réinitialisation de ICØR à -1.

d. 2 progression de 2 et test du compteur ordinal ; si sa vale dépasse 1999, il y a erreur.

d. 3 décodage de l'instruction contenue dans Mem (icor) et Mem (icor +1). Les différentes zones de l'instruction sont réparties dans ICØP, IRI, IAI, IAD.

d. 4 calcul de l'adresse effective, c'est-à-dire de l'indice de l'élément du tableau Mem où on trouve les 4 poids forts de l'opérande effectif. Ce calcul diffère selon le type de l'instruction en cours (manipulation d'index ou non). Il est détaillé au paragraphe suivant.

d. 5 aiguillage, selon le code opération, vers le traitement spécifique de l'instruction. Après le traitement des instructions de branchement on retourne en d. 3 ; on retourne en d. 2 après le traitement des autres instructions.

II. 2. 4 Calcul de l'adresse effective

L'adresse effective sera représentée par l'indice de l'élément du tableau Mem où se trouvent les poids forts de l'opérande effectif. On notera ief cet indice. Notons k l'entier 2 x iad + 1 : mem_k a pour valeur les poids forts du mot machine d'adresse iad. Dans le cas d'instructions où

l'indexation est impossible, on aura :

si iai = 0 ief = k

si iai = 1 ief = 2 x (mem_{k+1} + 10000 x mem_k) + 1.

Si l'indexation est possible, il faut ajouter à la valeur précédente de ief la quantité, 2 x ($indl_{iri}$ x 10000 + $ind2_{iri}$) dans le cas où iri est différent de 0.

II. 2. 5 Simulation des instructions

Chaque instruction MIP correspond à une séquence du programme de simulation. Le premier ordre FORTRAN de cette séquence est étiqueté par le code de l'instruction. Le dernier ordre (sauf dans le cas de l'instruction d'arrêt et des instructions de branchement) est : GØ TØ 101 (progression du compteur ordinal). Cette séquence est donc aisément repérable et modifiable.

II. 2. 5. 1 Simulation des instructions arithmétiques

Cette simulation suppose que les nombres réels de la machine utilisée par le programme de simulation comportent au moins 8 chiffres significatifs.

1 - Addition et soustraction

Le contenu de l'accumulateur est mis sous-forme du nombre réel :

$$r = 10^4 \times iacl + iac2$$

de même l'opérande effectif est mis sous la forme :

$$i = mem_{ief} \qquad j = mem_{ief+1}. \qquad s = 10^4 \times i + j$$

On effectue l'opération demandée sur ces 2 réels. Le résultat est un nombre réel Fortran : on donne à iac1 la valeur des 4 poids forts de ce résultat, et à iac 2 la valeur des 4 poids faibles.

2 - Multiplication

On détermine d'abord le signe du résultat à partir des signes des opérandes. On calcule ensuite la valeur absolue du résultat à partir des valeurs absolues des opérandes. On veut donc effectuer le produit :

$$p = (10^4 \times iacl + iac2) \times (10^4 \times i + j)$$

et obtenir le résultat sous la forme

$$p = 10^{12} \times iex1 + 10^8 \times iex2 + 10^4 iacl + iac2$$

c'est-à-dire avec 16 chiffres significatifs. L'emploi de réels n'ayant que 8 chiffres significatifs nécessite la décomposition suivante du calcul :

posons

$$\left.\begin{array}{l} r = iac2 \pm j \\ s = iac2 \pm i \\ t = iacl \pm j \\ u = iacl \pm i \end{array}\right\} \quad p = u \times 10^8 + (t + s) \times 10^4 + r$$

et décomposons ces nombre de la façon suivante :

$$\left.\begin{array}{l} r = 10^4\, i_1 + i_2 \\ s = 10^4\, j_1 + j_2 \\ t = 10^4\, k_1 + k_2 \\ u = 10^4\, \ell_1 + \ell_2 \end{array}\right\} \quad i_1,\ i_2,\ j_1,\ j_2,\ k_1,\ k_2,\ \ell_1,\ \ell_2 \text{ sont inférieurs à } 10^4 \text{ en valeur absolue.}$$

On a

$$iac2 = i_2$$
$$m1 = i_1 + j_2 + k_2$$
$$iacl = m1 - (m1 / 10000) \times 10000$$
$$m2 = m1 / 10000 + j\,1 + k\,1 + \ell_2$$
$$iex2 = m2 - (m2 / 10000) \times 10000$$
$$iex1 = m2 / 10000 + \ell_1$$

On affecte à iac2, iacl, iex2, iex1 le signe déterminé précédemment.

3 - Division

On peut effectuer la division du nombre

$$A = 10^{12} \times iex1 + 10^8\, iex2 + 10^4\, iacl + iac2$$

par le nombre

$$B = 10^4\, i + j$$

on veut mettre le quotient sous la forme

$$Q = 10^4\, iacl + iac2$$

et le reste sous la forme

$$R = 10^4\, iex1 + iex2.$$

Le quotient doit être plus petit que 10^8 en valeur absolue. Dans le cas contraire, on conservera les 8 chiffres de faibles poids de ce quotient. Le reste sera calculé à partir de ce quotient tronqué, et tronqué à son tour.

a) Calcul du quotient :

Soit les nombres réels

$$r = 10^4\, i + j$$
$$s = 10^4\, iacl + iac2$$
$$t = 10^4\, iex1 + iex2$$

On calcule $u = s/r$ et $v = t/r$ (division non entière)

Q = partie entière de u + partie entière de $(v \times 10^8)$

En principe v est strictement inférieur à 1. Tous les chiffres de la partie entière de $v \times 10^8$ sont donc significatifs.

On met Q sous la forme

$$Q = 10^4 \ell + m$$

où $|\ell| < 10^4$ et $|m| < 10^4$ et ℓ et u sont de même signe.

b) Calcul du reste

$$R = A - B \times Q$$
$$= 10^{12} \text{iex1} + 10^8 \text{iex2} + 10^4 \text{iac1} + \text{iac2} - (10^4 i + j)(10^4 \ell + m)$$
$$= 10^{12} \text{iex1} + 10^8 \text{iex2} - 10^8 i \times \ell - 10^4 i \times m - 10^4 j \times \ell + 10^4 \text{iac1} + \text{iac2} - j \times m.$$

On a $|R| < |B| < 10^8$ donc seuls nous intéressent les 8 chiffres de faibles poids de R.

Posons

$$\left.\begin{aligned} i \times m &= 10^4 i_1 + i_2 \\ j \times \ell &= 10^4 j_1 + j_2 \\ j \times m &= 10^4 k_1 + k_2 \end{aligned}\right\} \quad i_1, i_2, k_1, k_2 \text{ sont inférieurs à } 10^4 \text{ en valeur absolue}$$

R est de la forme

$$R = 10^4 \text{iac1} - 10^4 i_2 - 10^4 j_2 + \text{iac2} - 10^4 k_1 - k_2$$

On le met sous la forme

$$k = 10^4 \text{iex1} + \text{iex2}$$
$$\text{iac1} = \ell$$
$$\text{iac2} = m.$$

II. 2. 5. 2 Simulation des instructions d'entrées-sorties

Toutes les entrées-sorties se font en format 9A1.

En entrée, on passe par l'intermédiaire du sous-programme CØNV (voir II 2 3) qui met la valeur lue sous la forme de 2 entiers I et J, on met cette valeur dans le mot d'adresse effective :

MEM (IEF) = I

MEM (IEF+1) = J.

En sortie, il est nécessaire de mettre les 2 entiers représentant la valeur que l'on veut imprimer sous la forme d'une suite de 9 caractères.

En effet, si on imprime ces entiers en format In, les zéros de forts poids ne sont pas imprimés, et dans le cas d'une valeur négative, le signe apparait deux fois. Cette transformation est faite par l'intermédiaire du sous programme

ECR (I, J, IL)

où I et J sont les deux entiers et IL un tableau de 8 éléments où est rangé le résultat.

I et J sont supposés de même signe et inférieurs en valeur absolue à 10^4. En cas d'erreur, le signe considéré est celui de la somme de I et de J, et on imprime les 4 chiffres de faibles poids de chacun de ces nombres.

II. 2. 5. 3 Simulation des autres instructions

a) Instructions de branchement :

L'instruction de branchement inconditionnel (20) s'écrit de manière très simple :

20 ICØR = IEF

GØ TØ 92

l'étiquette 92 désigne le début du dépouillement de l'instruction contenue dans mem (icor), mem (icor+1).

Les instructions de branchement conditionnel se traitent alors de la manière suivante : si la condition est vérifiée branchement à 20 (traitement du branchement inconditionnel), sinon branchement à 101 (traitement de l'instruction suivante).

b) Manipulation de registres d'index

La simulation de ces instructions commence par un test pour savoir si la zone RI de l'instruction est bien différente de 0. Sinon, il y a diagnostic d'erreur et arrêt. Si oui on effectue l'opération demandée (chargement, rangement, addition) sur les deux nombres

10000 x mem (ief) + mem (ief + 1)

10000 x ind1 (iri) + ind2 (iri)

II. 3 Modifications possibles du programme de simulation

Rappelons qu'à chaque instruction de la machine MIP correspond une séquence du programme de simulation repérée par une étiquette identique au code de cette instruction. Il y a deux exceptions : à l'instruction de code 00 (chargement de l'accumulateur) correspond l'étiquette 34, à l'instruction de code 32 (rangement de registre d'index) correspond l'étiquette 31 (commune avec l'instruction 31).

La fin de chaque séquence est repérée par :

GØ TØ 92	pour les instructions de branchement
STØP	pour l'instruction d'arrêt (sur 10070)
GØ TØ 101	pour les autres instructions.

L'étiquette 106 correspond à une séquence du programme qui imprime le diagnostic d'erreur CO INCONNU et passe au programme suivant.

Branchement à ces différentes séquences ;

Le code de l'instruction est représenté par l'entier ICØP. Si ICØP > 33 on va à l'étiquette 106 : erreur. Sinon on calcule l'adresse effective et à la fin de ce calcul, on exécute un GØ TØ calculé dont la variable est K = ICØP + 1. Cet entier varie de 1 à 34. On aura donc 34 étiquettes dans ce GØ TØ calculé :

GØ TØ (34, 1, 2, 3, 106, 106, 106, 7, 106, 106, 10, . . . , 33), K

Pour supprimer une instruction, il suffit de remplacer dans ce GØ TØ calculé, l'étiquette qui lui correspond par 106.

Pour changer le code d'une instruction, il suffit de mettre, dans le GØ TØ calculé, l'étiquette qui lui correspond à la place de rang (le nouveau code +1) et de mettre 106 à son ancienne place.

Pour créer une nouvelle instruction, il suffit de lui affecter un code inutilisé et une étiquette, de mettre cette étiquette au rang (le code +1) dans le GØ TØ calculé, et d'écrire la séquence correspondante. Toutefois, il faut savoir que :

1) l'opérande effectif est représenté par mem (ief) et mem (ief + 1)

2) le passage à l'instruction MIP suivante se fait dans le programme de simulation par GØ TØ 101

3) un code opération supérieur ou égal à 30 supprime l'indexation dans le calcul de l'adresse effective.

ANNEXE 1

CODE ET DESCRIPTION DES INSTRUCTIONS DE MIP

Notations :

L'adresse effective d'une instruction est l'adresse déduite de la zone adresse en effectuant, s'il y a lieu, l'adressage indirect et l'indexation.

L'opérande effectif est le contenu du mot d'adresse effective.

On désigne le contenu d'un mot par ; (adresse du mot)

Codes	Description
00	Chargement de l'accumulateur L'opérande effectif est mis dans l'accumulateur. L'extension est remise à zéro. (adresse effective) → Accu 0 → Ext
01	Chargement de l'extension L'opérande effectif est mis dans l'extension. (adresse effective) → Ext
02	Rangement de l'accumulateur Le contenu de l'accumulateur est rangé dans le mot d'adresse effective (Accu) → Adresse effective.
03	Rangement de l'extension. Le contenu de l'extension est rangé dans le mot d'adresse effective. (Ext) → Adresse effective
07	Arrêt Cette instruction est la dernière exécutée du programme. L'adress qui suit est inutile, mais doit être présente. Cette instruction provoque l'impression de 'FIN' et le passage au programme suivant.

10 Addition

L'opérande effectif est ajouté au contenu de l'accumulateur. Le résultat est dans l'accumulateur :

(Accu) + (adresse effective) → Accu

11 Soustraction

On soustrait l'opérande effectif du contenu de l'accumulateur. Le résultat est rangé dans l'accumulateur.

(Accu) - (Adresse effective) → Accu

14 Multiplication

Le contenu de l'accumulateur est multiplié par l'opérande effectif. Le résultat est rangé dans l'extension (8 poids forts) et dans l'accumulateur (8 poids faibles). Le signe est présent dans l'extension et dans l'accumulateur.

(Accu) x (adresse effective) → Ext et Accu

15 Division

Le nombre dont les 8 poids forts sont dans l'extension et les 8 poids faibles dans l'accumulateur est divisé par l'opérande effectif. Le quotient est dans l'accumulateur. Le reste est dans l'extension $[10^8 \times (\text{Ext}) + (\text{Accu})] / (\text{adresse effective}) \rightarrow \text{Accu}$.

16 Lecture d'un mot sur carte

La valeur à lire (1 signe et 8 chiffres décimaux) est perforée à partir de la 1ère colonne.

Elle est rangée à l'adresse effective

valeur lue → Adresse effective.

17 Ecriture d'un mot sur machine à écrire

L'opérande effectif est imprimé sur machine à écrire.

18 Branchement à l'adresse effective si le contenu de l'accumulateur est strictement positif. Sinon on va en séquence.

19 Branchement à l'adresse effective si le contenu de l'accumulateur est nul, sinon on va en séquence.

20 Branchement inconditionnel à l'adresse effective.

Pour toutes les instructions suivant l'adressage indexé est impossible.

30 Mise à zéro de registre d'index

Le registre d'index indiqué dans la zone RI de l'instruction prend la valeur 0.

Adressage indexé interdit.

31 Chargement de registre d'index.

L'opérande effectif est mis dans le registre d'index indiqué dans la zone RI de l'instruction.

(adresse effective) → RX designé.

32 Rangement de registres d'index

Le contenu du registre d'index indiqué dans la zone RI de l'instruction est rangé dans le mot d'adresse effective.

(Rx désigné) → adresse effective.

33 Modification de registre d'index

L'opérande effectif est ajouté au contenu du registre d'index indiqué dans la zone RI de l'instruction. Le résultat est rangé dans ce registre d'index.

Rxi ← (Rx i) + (adresse effective)

CHAPITRE III

DESCRIPTION DES MACHINES SIMULEES ILM70

INTRODUCTION - PRESENTATION

ILM70-1, ILM70-2 et ILM70-3, sont 3 machines fictives de complexité croissante. Leurs langages machines sont des sous-ensembles du langage machine CII 10070 et, pour que le passage d'une machine à l'autre ne pose pas de problème, nous avons défini ces machines de telle sorte que le jeu d'instructions de ILM70(i-1) soit inclus dans celui de ILM70(i).

Cependant, il a été nécessaire de faire une exception pour la première machine qui doit être une machine très simple, ne présentant aucune difficulté dans le maniement des instructions elles-mêmes. Or les instructions de branchements sur 10070 se font par produit logique d'une zone de l'instruction avec le code de condition. Ceci risquerait de dérouter les personnes peu familiarisées avec les nombres binaires. Les instructions de branchement de ILM70-1 ne seront donc pas des instructions du 10070, mais des branchements conditionnels simples (branchement si registre positif par exemple). On ne les retrouvera pas dans ILM70-2 où elles seront remplacées par les instructions du 10070.

ILM70-1 est la machine d'initiation. Il faudra donc apporter beaucoup de soin à sa présentation qui devra être progressive.

Par exemple, dans un premier temps, on ne parlera pas d'entrées-sorties : les données seront implantées en mémoire en même temps que le programme, et après l'exécution de chaque programme, on effectuera un "dump" des registres et de la mémoire, ce qui permettra de vérifier les résultats.

Lors de l'écriture des premiers programmes on n'utilisera que 2 registres et on se limitera à des programmes n'utilisant pas d'instructions de branchement.

On trouvera l'exposé de cette présentation au chapitre suivant.

III. 1 ILM70-1 - Description

ILM70-1 permet une arithmétique entière uniquement. Les nombres négatifs sont représentés en complément à 2. Ses caractéristiques sont les suivantes :

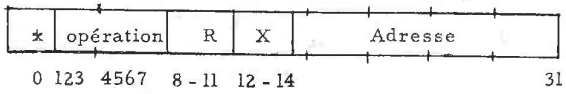
III. 1. 1 Une mémoire centrale formée de 4096 mots banalisés de 32 bits dont les adresses varient de 0 à 4095.

III. 1. 2 16 registres généraux qui sont les 16 premiers mots de la mémoire. Ils servent tous d'accumulateur entier. Les registres 1 à 7 sont aussi des index.

III. 1. 3 2 registres internes non adressables : un compteur ordinal et un registre - instruction. Le compteur ordinal apparait en tête des "dump" mémoire.

III. 1. 4 2 périphériques : 1 lecteur de cartes et 1 imprimante. Les opérations d'entrées-sorties sont commandées par une unité d'échange, l'unité centrale ne jouant qu'un rôle mineur dans l'exécution de ces opérations. Cette distinction intervient peu pour ILM70-1 et ILM70-2, les opérations d'E/S étant assurées par des sous programmes de service.

III. 1. 5 Le format des instructions est le même que sur 10070



On y trouve de la gauche vers la droite :

* dans le bit 0 la marque d'adressage indirect

- dans les bits 1 à 7 le code de l'opération à effectuer

- dans les bits 8 à 11 le numéro du registre général R qui interviendra dans l'opération

- dans les bits 12 à 14 on peut spécifier comme index l'un des registres 1 à 7. Si X = 0, il n'y a pas d'indexation.

- dans les bits 15 à 31 une adresse qui après modification éventuelle par adressage indirect et (ou) indexation permet de repèrer l'opérande.

L'instruction sera représentée par une suite de 8 chiffres hexadécimaux.

III. 1. 6 Jeu d'instructions.

Notations :

- A est l'adresse effective obtenue après un adressage indirect et / ou une indexation eventuels.

- (A) est le contenu du mot d'adresse A

- R est le numéro de registre indiqué dans les bits 8 à 11 de l'instruction.

- RU1 est le numéro de registre obtenu en faisant la somme logique des bits 8 à 11 de l'instruction et de 1 : Si R est pair, RU1 = R + 1
Si R est impair RU1 = R.

DESCRIPTIONS DES INSTRUCTIONS

code hexadécimal	mnémonique	Description
32	LW	Chargement mot : (A) → R
35	STW	Rangement mot : (R) ⟶ A
30	AW	Addition mot : (R) + (A) → R
38	SW	Soustraction mot : (R) - (A) → R
37	MW	Multiplication mot : (RU1) * (A) ⟶ R, RU1 Le résultat de la multiplication est un nombre binaire de 64 bits. Si R est pair, on trouve dans R et R+1 les 64 chiffres binaires du résultat. Si R est impair on trouve dans R les 32 chiffres binaires de faibles poids du résultat
36	DW	Division mot (R, RU1) / (A) ⟶ RU1 reste ⟶ R Si le reste n'est pas nul, il a le même signe que le dividende (contenu initial de R) Si R est impair, la division effectuée est (R) / (A) → R et le reste est perdu.

66	BRN	Branchement si registre nul : Si (R) = 0 branchement en A sinon passage en séquence.
67	BRP	Branchement si registre positif Si (R) > 0 branchement en A sinon passage en séquence.
68	BRI	Branchement inconditionnel en A
6A	BAL	Branchement à un sous-programme Si R = F_{16}, si A = 1000_{16} branchement au sous-programme de lecture et conversion Si R = F_{16}, si A = 1100_{16} branchement au sous-programme de conversion et d'écriture. Sur ILM70-1 toute autre adresse de sous programme provoquera un diagnostic d'erreur 'Adresse incorrecte' et l'abandon du programme.
04	CALl	Arrêt : La dernière instruction exécutée doit être '04900001'$_{16}$

III. 1. 7 Sous-programmes de lecture et d'écriture

Sur ILM70-1, le programmeur dispose de 2 sous-programmes d'entrée-sortie qui ont été implantés dans une extension de la mémoire. Ces sous programmes utilisent comme registre de liaison le registre 15 qui est donc modifié.

- Le sous-programme de lecture et de conversion a été implanté à l'adresse hexadécimale 1000. Ce sous-programme lit sur une carte un nombre décimal cadré entre la première colonne et le premier blanc. Ce nombre est converti en binaire et le résultat est rangé dans le registre 10.

- Le sous-programme d'écriture et de conversion convertit en décimal le contenu du registre 10 et imprime le résultat de cette conversion.

III. 1. 8 Remarque générale :

A de nombreux égards le format de l'instruction décrit en 1. 5 ne se justifie pas pour une telle machine (il suffirait de 12 bits pour la partie adresse, 4 pour le code opération), tout au moins si on le retire de son contexte : introduction à des machines plus élaborées et mieux dotées qui conduiront au 10070. On pourra insister sur ce point lors de la présentation.

III. 2 ILM70-1 Fonctionnement

III. 2. 1 Chargement du programme

Le programme est chargé à partir de l'adresse 16 (10 en hexadécimal). Il est lu sur cartes, à raison d'une instruction par carte perforée dans les colonnes 1 à 8. Le paquet de cartes doit se terminer par une carte !EØD. Lorsque cette carte est lue le compteur ordinal est initialisé à 10_{16}, et l'exécution commence selon l'un des 2 modes de fonctionnement :

Fonctionnement ininterrompu - cf (III. 2. 3)

Fonctionnement en pas à pas - cf (III. 2. 4).

III. 2. 2 Post Mortem Dump

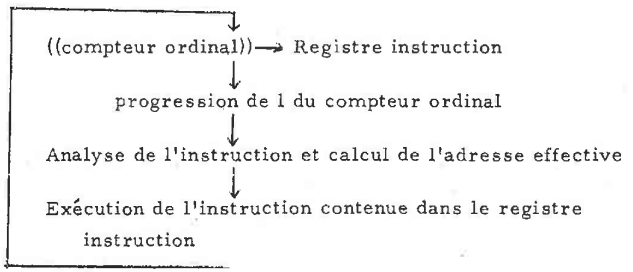
A la fin de l'éxecution d'un programme, on peut demander l'impression du contenu des registres, du compteur ordinal et de la mémoire. On utilisera pour cela la carte de commande suivante (il s'agit d'une commande du système BPM 10070)

! PMDI (CØR, CØR), (MEM, MEM + FFF)

Cette carte se place entre l'appel du programme de simulation et le programme ILM70. Un exemple de programme ILM-70 auquel ont été adjointes les cartes de commande nécessaires à son passage est donné page suivante.

III. 2, 3 Fonctionnement ininterrompu

Il se déroule selon la boucle suivante



Cette boucle n'est interrompue que par une instruction d'arrêt ou la détection d'une anomalie (voir III. 2. 5).

Pour obtenir ce mode de fonctionnement, on doit présenter le programme ILM70 sous la forme suivante :

```
!JØB        n° de compte, nom, priorité.
!RUN        (LMN, NIV1, TMAPRG00)
!PMDI       (CØR, CØR), (MEM, MEM+FFF)
!DATA
6AF01000    CECI EST UN EXEMPLE DE PROGRAM-
3230000A    ME ILM-70 PRET AU PASSAGE.
6AF01000    LES INSTRUCTIONS SONT PERFOREES
3030000A    A RAISON DE UNE PAR CARTE, A PAR-
6AF01000    TIR DE LA 1ERE COLONNE - CE QUI EST
3720000A    AU DELA DE LA COLONNE 8 EST
3530000A    CONSIDERE COMME COMMENTAIRE.
6AF01100    CE PROGRAMME LIT 3 NOMBRES A, B, C CALCU-
04900001    LE (A+B) x C ET IMPRIME LE RESULTAT
!EØD
+12
-3
100
```

Le "post mortem dump" a été demandé.

III. 2. 4 Fonctionnement en "pas à pas"

Ce mode d'exécution permet de suivre l'exécution du programme. Il se déroule suivant la même boucle que précédemment. En outre, il a pour effet de faire imprimer le contenu des registres, le contenu du compteur ordinal et le contenu de la mémoire chaque fois que l'exécution d'une instruction se termine.

Pour demander ce mode de fonctionnement, il suffit d'ajouter juste après la carte !RUN de l'exemple précédent la carte suivante :

```
!SNAP   PASAP, PASAPAS, (CØR, CØR), (MEM, MEM+FFF).
```

III. 2. 5 Diagnostics d'erreurs

Certaines anomalies provoquent l'impression d'un diagnostic d'erreur et l'abandon du programme.

Ces diagnostics sont les suivants :

- 'ERREUR A LA LECTURE' au chargement, une carte contenait autre chose qu'un chiffre hexadécimal dans les 8 premières colonnes.

- 'ADRESSE INCØRRECTE' lors de l'exécution on a rencontré une adresse non comprise entre 0 et 4095.

- 'CODE INSTRUCTION INTERDIT' la zone opération du mot qui vient d'être chargé dans le registre instruction ne correspond pas à une instruction de ILM70-1.

- 'ENTIER INCORRECT' ce diagnostic est imprimé si, lors d'une exécution du sous-programme de lecture
  - on a trouvé, sur la carte lue, autre chose qu'un chiffre décimal ou un signe ;
  - l'entier lu est supérieur en valeur absolue à $2^{31}-1$.

III. 3 ILM70-2 Description

III. 3.1 Introduction

ILM70-2 ne permet encore qu'une arithmétique entière par mot. Son champ d'application est donc identique à celui de ILM70-1. Les progrés porteront sur le jeu d'instructions : celles-ci moins "élémentaires". On retrouve ici la même progression que pour les machines réelles, pour lesquelles, la pratique de la programmation ayant permis une meilleure définition des besoins, les progrès dans la réalisation du "hardware" permettant de satisfaire ces besoins, on a vu introduire des instructions de plus en plus élaborées. C'est le cas, par exemple des instructions de branchement avec incrémentation de registre.

Pour permettre une plus grande variété de tests, on a ajouté à l'unité centrale un registre interne de 4 bits appelé code de condition. Ce registre est positionné après l'exécution de chaque instruction (ou presque). Il indique comment s'est déroulée cette exécution, et donne certains renseignements sur le résultat.

Les différences entre les jeux d'instructions d'ILM70-1 et d'ILM70-2 sont donc les suivantes :

les instructions arithmétiques et de transfert modifient le code de condition lors de leur exécution, mais leur effet reste inchangé par ailleurs. Les instructions de branchement BRP et BRN sont remplacées par des instruction de branchement selon le contenu du code de condition. On a ajouté une instruction de comparaison, une instruction de test d'un mot en mémoire, des instructions de bouclage (branchement avec incrément et décrément de registre).

On a étendu le rôle de l'instruction BAL (appel de sous-programme). Les décalages sont également introduits pour établir un travail à l'intérieur du mot.

D'autre part 4 instructions à adressage immédiat (chargement, addition, multiplication, comparaison) permettent de manipuler plus aisément les constantes.

III. 3. 2 Code de condition :

C'est un registre interne de 4 bits. Il est mis à jour automatiquement à la fin de presque chaque instruction en indiquant comment celle-ci s'est exécutée. Il donne des renseignements sur le résultat obtenu. Il est utilisé par les instructions de branchement. Il n'est pas accessible au programmeur sur ILM70-2.

III. 3. 3 Instructions arithmétiques et de transfert :

Elles sont définies comme pour ILM70-1. De plus, elles modifient le code condition de la façon suivante :

32 LW Chargement mot :

Les positions 3 et 4 du code de condition sont modifiées :

Valeur du CC	Résultat dans R
- - 0 0	nul
- - 0 1	négatif
- - 1 0	positif

35 STW Rangement mot :

Le code de condition n'est pas modifié.

30 AW Addition mot :

Les 4 positions du code de condition sont modifiées.

Valeur du CC	Résultat dans R
- - 0 0	nul
- - 0 1	négatif
- - 1 0	positif
- 0 - -	pas de débordement
- 1 - -	débordement
0 - - -	pas de report au delà du chiffre binaire 0
1 - - -	report au delà du chiffre binaire 0

38 SW Soustraction mot :

Les 4 positions du code de condition sont modifiées de la même manière qu'après l'instruction AW.

37 MW Multiplication mot :

3 positions du code de condition sont modifiées :

Valeur du CC	Résultat dans R, RU1
- - 0 0	nul
- - 0 1	négatif
- - 1 0	positif
- 0 - -	La partie significative du résultat est contenue dans RU1
- 1 - -	Le résultat déborde la capacité du seul registre RU1

36 DW Division mot :

3 positions du code de condition sont modifiées :

Valeur du CC	Résultat dans RU1
- 0 0 0	quotient nul, pas de débordement
- 0 0 1	quotient négatif, pas de débordement
- 0 1 0	quotient positif, pas de débordement
- 1 0 0	débordement virgule fixe.

III. 3. 4 Instructions de comparaison

31 CW Comparaison de mot

Le contenu du registre R est comparé à l'opérande effectif, les contenus de ces 2 mots étant traités comme des grandeurs arithmétiques en virgule fixe.

Le code de condition est positionné en fonction du résultat de la comparaison

Valeur du CC	Résultat de la comparaison
- - 0 0	égalité
- - 0 1	(R) < (A)
- - 1 0	(R) > (A)
- 0 - -	(R) ∩ (A) = 0
- 1 - -	(R) ∩ (A) ≠ 0

33 MTW Modification et test mot

La zone R est considérée comme un nombre positif, négatif, ou nul en complément à 2 sur 4 bits. Elle contient le "modificateur". Le premier bit de la zone R (bits 8 à 11) de l'instruction est étendu de 28 positions vers la gauche. Le nombre binaire obtenu est ajouté à l'opérande effectif. La somme est rangée à l'adresse effective

$$(A) + R \longrightarrow A$$

Le code de condition est fixé en fonction du résultat

Valeurs du CC	Résultat dans A
- - 0 0	nul
- - 0 1	négatif
- - 1 0	positif
- 0 - -	pas de débordement virgule fixe
- 1 - -	débordement vigule fixe
0 - - -	pas de report pour le mot
1 - - -	report pour le mot

Si la zone R (bits 8 à 11) de l'instruction est nulle, l'instruction consiste simplement en un test de l'opérande effectif pour indiquer s'il est nul, positif ou négatif.

III. 3. 5 Instructions de branchement

68 BCR Branchement sur condition zéro

Si le produit logique du code de condition et des bits 8 à 11 de l'instruction (c. a. d. R) est égal à 0, branchement en A, sinon passage en séquence.

Si les bits 8 à 11 de l'instruction sont nuls, le branchement a toujours lieu et BCR est équivalente à une instruction de branchement inconditionnel.

69 BCS Branchement sur condition 1

Si le produit logique du code de condition et des bits 8 à 11 de l'instruction est différent de 0, branchement en A sinon passage en séquence.

Si les bits 8 à 11 de l'instruction sont nuls (BCS, 0) l'instruction est alors sans effet (pas d'opération).

III. 3. 6 Instructions de "bouclages" (ou d'itérations)

Ce sont encore des instructions de branchement.

65 BIR Incrément registre et branchement

Le contenu du registre R est augmenté de 1. Si la nouvelle valeur de R est négative , il y a branchement à l'adresse effective sinon on passe en séquence :

$(R) + 1 \longrightarrow R$

si $(R) < 0$ branchement en A

sinon on va en séquence.

64 BDR Décrement registre et branchement

Le contenu du registre R est diminué de 1. Si la nouvelle valeur de R est positive il y a branchement à l'adresse effective, sinon on passe en séquence ;

$(R) - 1 \longrightarrow R$

Si $(R) > 0$ Branchement en A sinon on passe en séquence.

III. 3. 7 Branchement à un sous-programme

L'instruction BAL (branchement à un sous-programme) n'est plus limitée, comme sur ILM70-1, à l'utilisation des sous-programmes de lecture et d'écriture. Elle peut être utilisée pour se brancher à n'importe quel sous-programme. Sa description est la suivante :

6A BAL Branchement et enchainement

L'adresse de l'instruction +1 est chargée dans le registre R

On éxécute alors un branchement à l'adresse effective.

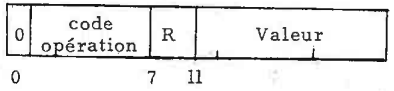
Les sous-programmes de lecture et d'écriture sont toujours utilisables. Rappelons qu'ils sont respectivement implantés aux adresse hexadécimales 1000_{16} et 1100_{16}.

III. 3, 8 Décalage

Cette instruction est identique à l'instruction de décalage (S) du 10070. On trouvera la description de cette instruction dans la notice de la CII, pages III. 82 à III. 85. Ces pages sont données en annexe à la fin du chapitre.

III. 3. 9 Instructions à adressage immédiat :

Le format de ces instructions est le suivant



L'opérande est le nombre binaire de 32 bits obtenu en étendant le signe de la zone "Valeur" de l'instruction de 12 positions vers la gauche.

L'utilisation de l'adressage indirect et de l'indexation est interdite pour ces instructions.

code hexadécimal	mnémonique	description :
22	LI	Chargement immédiat Valeur avec signe étendu → R Le positionnement du code de condition est le même que pour l'instruction LW.
20	AI	Addition Immédiate $(R)_+$ Valeur avec signe étendu → R le positionnement du code de condition est le même que pour l'instruction AW.
23	MI	Multiplication Immédiate $(RU1)_*$ Valeur avec signe étendu → R et RU1 le positionnement du code de condition est le même que pour l'instruction MW,
21	CI	Comparaison Immédiate Le contenu du registre R est comparé à la valeur avec signe étendu. Le code de condition est positionné en fonction du résultat de la comparaison, et de la même manière que pour l'instruction CW.

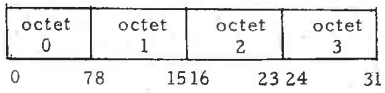
III. 4 ILM70-3 - Description

III. 4. 1 Introduction

Le répertoire des instructions de cette machine comprend toutes les instructions de ILM70-2. Ces instructions permettent uniquement de manipuler des mots. Dans les problèmes à résoudre on a souvent besoin de traiter des informations de formats divers. On est amené à diviser la mémoire en "cellules" de différentes tailles, ce qui pose des problèmes d'adressage. Dans un premier temps, on ajoutera donc aux instructions de ILM70-2 des instructions de manipulation d'octets et de chaines d'octets. Dans un second temps, on parlera de caractères, d'entrées-sorties, et on utilisera des instructions d'entrées-sorties simplifiées, et non plus de sous-programmes.

III. 4. 2 Adressage par octet

Un mot mémoire de ILM70-3 peut être divisé en 4 octets :



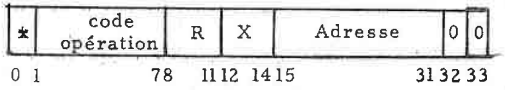
mot mémoire ILM70-3

L'adresse de l'octet i du mot d'adresse Ad sera 4 x Ad + i. Pendant l'exécution d'instruction de manipulation d'octets, l'objet désigné doit être un octet. On appellera cet octet "octet effectif".

Le type d'adressage employé est le même que sur 10070. Dans une instruction 10070, la partie adresse comprend 17 bits. Elle permet donc d'adresser directement 128 k "cellules". Or la mémoire centrale du 10070 peut contenir jusque 128 K mots, c.a.d. 512 K octets. On ne pourrait donc pas adresser directement tous les octets de la mémoire avec 17 bits. En conséquence l'adresse contenue dans les bits 15 à 31 d'une instruction 10070 est toujours une adresse de mot[1]. Il en sera de même sur ILM70-3, puisque cette machine est destinée à faciliter l'apprentissage de la programmation sur 10070, mais cela ne se justifie pas si l'on retire ILM70 de ce contexte puis-

A l'exécution, il faut donc convertir cette adresse de mot en adresse d'octet. Pour cela il a été prévu dans le registre instruction une zone adresse de 19 bits, les 2 derniers bits de cette zone étant mis à zéro (voir figure).

[1] Remarque : il nous semble maladroit de dire, comme sur le "manuel d'utilisation 10070" (NT 2444) que ces instructions ont un adressage par octet : c'est faux dans leur écriture (puisque c'est toujours une adresse de mot qui apparait dans l'instruction). C'est vrai pendant leur exécution, mais aussi pendant l'exécution de toute instruction 10070 (l'adresse effective est toujours une adresse d'octet quelque soit le type d'adressage).



registre instruction d'ILM70

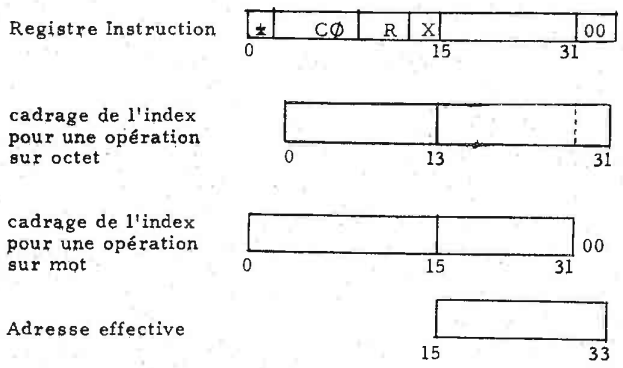
Une instruction utilisée ainsi permet donc d'adresser l'octet 0 du mot designé par la zone adresse.

Pour désigner d'autres octets on aura recours à l'indexation : d'une manière générale, sur ILM70, si le registre d'index désigné par une instruction contient un entier k, il permet de repèrer le kième élément à partir de l'adresse contenue dans le registre instruction (celle-ci peut avoir été éventuellement modifiée par adressage indirect). Ceci est indépendant du type d'information traité par l'instruction.

Lorsque l'indexation est utilisée, le contenu du registre d'index est cadré automatiquement par rapport au registre instruction selon le type d'information manipulée par l'instruction.

S'il s'agit d'octets, chiffre binaire de plus faible poids du contenu de l'index est aligné avec le chiffre binaire 33 du registre instruction.

S'il s'agit de mots, il est aligné avec le chiffre binaire 31 du registre instruction. Une adresse de 19 chiffres binaires est alors obtenue par addition.



Donc, lors d'une instruction de manipulation d'octets, si Ad est l'adresse de mot contenue dans les bits 15 à 31 de l'instruction, et si X est le n° du registre d'index éventuel contenu dans les bits 12 à 14 de l'instruction l'adresse de l'octet effectif vaut :

Cas 1 ni adressage indirect, ni indexation :

adresse effective = 4 x Ad

L'octet effectif est l'octet 0 du mot d'adresse Ad.

Cas 2 adressage indirect, pas d'indexation :

adresse effective = 4 x (Ad)

L'octet effectif est l'octet 0 du mot d'adresse (Ad).

Cas 3 pas adressage indirect, indexation :

adresse effective = 4 x Ad + (X).

Si X contient l'entier k, l'octet effectif est le kième à partir de l'octet 0 du mot d'adresse Ad.

Cas 4 Adressage indirect et indexation :

adresse effective = 4 x (Ad) + (X).

Si X contient l'entier k, l'octet effectif est le kième à partir de l'octet 0 du mot d'adresse (Ad).

III. 4. 3 Instructions de manipulation d'octets

Ces instructions sont des instructions 10070 à "adressage par octet".

72 LB Chargement octet

octet effectif → octet 3 de R (bits 24 à 31)

Les positions 0 à 23 de R sont remises à 0.

Positionnement du code de condition

1 2 3 4	Résultat dans R
- - 0 0	nul
- - 1 0	différent de 0.

75 STB Rangement octet

octet 3 de R (bits 24 à 31) → adresse effective

R est inchangé.

71 CB Comparaison octet

Cette instruction compare l'octet effectif et l'octet 3 de R (bits 24 à 31).

Le code de condition est positionné en fonction du résultat de la comparaison :

1 2 3 4	Résultat de la comparaison
- - 0 0	égalité
- - 0 1	$(R)_{24-31}$ < octet effectif
- - 1 0	$(R)_{24-31}$ > octet effectif
- 0 - -	$(R)_{24-31}$ ∩ octet effectif = 0
- 1 - -	$(R)_{24-31}$ ∩ octet effectif ≠ 0

73 MTB Modification et test d'octet

Si la zone R (bits 8-11) est nulle l'instruction MTB consiste en un test de l'octet effectif dont le résultat est dans le code de condition.

Sinon, le premier bit de la zone R (bit 8 de l'instruction) est étendu de 4 positions vers la gauche. L'octet obtenu est ajouté à l'octet effectif. Le code de condition est positionné en fonction du résultat de cette addition

1 2 3 4	
- 0 0 0	nul
- 0 1 0	différent de zéro
0 - - -	pas de report
- - - -	report.

III. 4. 4 Instructions de manipulation de chaines d'octets

Elles représentent par rapport aux instructions du paragraphe précédent la même évolution que celle décrite en III. 3. 1.

Ce sont en effet des instructions perfectionnées, permettant une programmation plus facile lorsque l'on manipule des chaines de caractères.

Leur description, que l'on trouvera en annexe à la fin du chapitre est celle donnée dans le "Manuel d'exploitation 10070" (réf NT 2444) p. III-100 à III-115.

III. 4. 5 Instructions d'entrées-sorties de ILM70-3

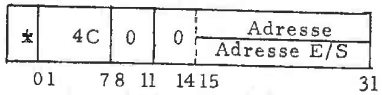
ILM70 possède 2 organes périphériques : un lecteur de cartes et une imprimante.

Tous les échanges d'informations entre périphériques et mémoire sont commandés par une unité d'échange, ce qui permet à l'unité centrale de travailler indépendamment de l'opération d'entrée -sortie en cours.

Les renseignements sur l'opération d'E/S à effectuer sont fournis à l'unité d'échange sous forme de commandès. Le format de ces commandes est de 2 mots mémoires. L'adresse du ler mot est forcément paire. Les instructions d'ILM70 relatives aux entrées-sorties sont les suivantes :

a) 4C SIØ Départ entrées-sorties

Format de l'instruction



Les bits 8 à 14 doivent être toujours à 0.

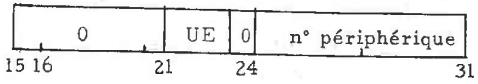
On trouve dans les bits 15 à 31

- s'il y a adressage indirect, une adresse de mot.

On trouvera dans ce mot une adresse d'entrée -sortie

- s'il n'y a pas adressage indirect une adresse d'entrée -sortie.

Une adresse d'entrée -sortie est de la forme :



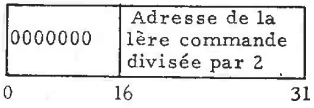
Sur ILM70, il n'existe qu'une unité d'échange dont le code est 0.

Les numéros de périphériques sont :

3 pour le lecteur de cartes

2 pour l'imprimante.

L'instruction SIØ utilise le registre 0 qui doit contenir l'adresse de la première commande d'unité d'échange. Cette adresse (forcément paire est cadrée comme suit :



Deux jeux de commandes ont été écrits et implantés dans l'extension de la mémoire ILM70-3. Ils correspondent aux deux opérations :

a) jeu de commandes implantées à partir de l'adresse 2000_{16} : lecture de 80 caractères sur cartes en utilisant comme tampon les mots d'adresse FEC_{16} à FFF_{16}.

b) jeu de commandes implanté à partir de l'adresse $20A0_{16}$: impression sur imprimante des 132 caractères contenus dans les mots d'adresse FCB_{16} à FEB_{16}.

Le code de condition est modifié par l'instruction SIØ :

Valeur du Code	Déroulement de SIØ
1 2 3 4	
- 0 - -	Départ réussi
- 1 - -	Départ non réussi
0 - - -	Adresse E/S reconnue
1 - - -	Adresse E/S non reconnue.

En cas de 'départ non réussi' ou de 'adresse E/S non reconnue' il y a en plus du positionnement du code de condition, impression de diagnostics explicites. Le programme se poursuit en séquence.

Dans le cas où l'adresse de commande précisée dans le registre 0 ne correspond pas à un des 2 jeux standard présentés ci-dessus, ou s'il y a incompatibilité du jeu de commandes et du périphérique désigné, l'unité d'échange enverra une interruption qui provoquera l'exécution d'un sous-programme implanté dans l'extension de la mémoire. Ce sous-programme

imprime le diagnostic

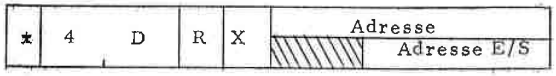
'COMMANDE UE INCORRECTE'

et provoque l'abandon du programme ILM70 en cours.

Après l'instruction SIØ on poursuit en séquence l'exécution du programme. L'opération d'entrée -sortie est exécutée simultanément. On peut tester la fin de l'entrée -sortie ou l'état du périphérique par l'instruction suivante :

TIO X '4D' Test entrée -sortie .

Adressage par mot :



R doit toujours être nul.

Le code de condition est positionné :

1 2 3 4	
0 - - -	Adresse d'E/S reconnue
1 - - -	Adresse d'E/S non reconnue
- 0 - -	Une SIØ peut être exécutée avec succés
- 1 - -	Une SIØ ne peut être exécutée avec succés.

ANNEXE

INSTRUCTIONS DU CII 10070

DECALAGE, DEPLACEMENT DE CHAINES D'OCTETS

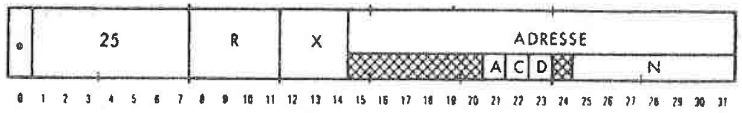
COMPARAISON DE CHAINES D'OCTETS

■ DECALAGE

Le format d'instruction pour les opérations de décalage logique circulaire, et arithmétique, est la suivant :

SHIFT

Décalage
(adressage par mot)



Si l'instruction de décalage est utilisée sans indexation ni adressage indirect, les c. b. 21 à 31 de la zone adresse déterminent le genre, la direction et la valeur du décalage. Si l'adressage indirect est utilisé, les c. b. 15 à 31 de l'instruction sont utilisés pour déterminer le mot indir dans lequel les c. b. 21 à 31 déterminent le genre, la direction et la valeur du décalage.

Si l'indexation est seule utilisée, les c. b. 21 à 23 du mot instruction déterminent le genre du décalage; la direction et la valeur du décalage sont déterminées par les c. b. 25 à 31 de l'instruction auxquels sont ajoutés les c. b. 25 à 31 du registre d'index désigné.

Si l'adressage indirect et l'indexation sont utilisés simultanément, les c. b. 15 à 31 de l'instruction sont utilisés pour accéder au mot indirect, et les c. b. 21 à 23 du mot indirect déterminent alors le genre de décalage; la direction et la valeur du décalage sont déterminées par les c. b. 25 à 31 du mot indirect auxquels on ajoute les c. b. 25 à 31 du registre d'index désigné.

Les c. b. 21, 22, et 23 de l'adresse effective déterminent le type de décalage, de la façon suivante :

Positions de c.b. 21	22	23	Genre de décalage
0	0	0	Logique, registre unique
0	0	1	Logique, double registre
0	1	0	Circulaire, registre unique
0	1	1	Circulaire, double registre
1	0	0	Arithmétique, registre unique
1	0	1	Arithmétique, double registre
1	1	0	non défini
1	1	1	non défini

Les c. b. 25 à 31 de l'adresse effective déterminent la direction et la valeur du décalage

Ce nombre (N) est traité comme un entier algébrique binaire de 7 c. b., le c. b. de poids le plus fort (c. b. 25) représentant le signe (les entiers négatifs sont représentés en complément à 2).

Un nombre positif de décalages provoque un décalage vers la gauche de N positions de c. b. Un nombre négatif de décalages provoque un décalage vers la droite de N positions de c. b. La valeur de N est bornée de la façon suivante :

$-64 \leqslant N \leqslant +63$

Toutes les opérations de décalage d'un double registre nécessitent que la zone R de l'instruction soit paire et traitent les registres R et Ru1 comme un registre unique de 64 c. b., le c. b. de plus fort poids (c. b. 0 du registre R) représentant le signe de l'ensemble du double registre. Si la zone R de l'instruction S est impaire et si une opération de décalage d'un double registre est demandée, le double-mot de registre est obtenu en utilisant deux fois le contenu du registre R; pour les 32 c.b. de fort poids et pour les 32 c. b. de faible poids du double-mot.

L' opération de décalage est ensuite effectuée et les 32 c. b. de fort poids du résultat sont chargés dans le registre R.

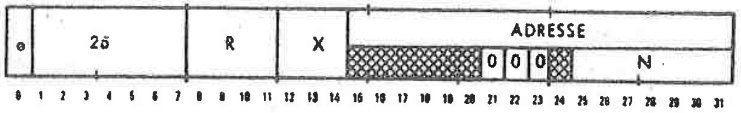
Un débordement survient (uniquement dans les décalages vers la gauche), chaque fois que la valeur du c. b. de signe (c. b. 0 du registre R) est modifiée. Lors de l'achèvement d'un décalage logique à gauche, circulaire à gauche et arithmétique à gauche, le code de condition prend les valeurs suivantes :

1	2	3	4	Résultat du décalage
0	-	-	-	nombre pair de 1 ayant traversé l'extrémité gauche du registre R
1	-	-	-	nombre impair de 1 ayant traversé l'extrémité gauche du registre R
-	0	-	-	pas de débordement pour un décalage à gauche
-	1	-	-	débordement pour un décalage à gauche

Lors de l'achèvement d'un décalage logique à droite, circulaire à droite et arithmétique à droite, le code de condition prend la valeur suivante :

1	2	3	4
0	0	-	

Décalage logique, registre unique

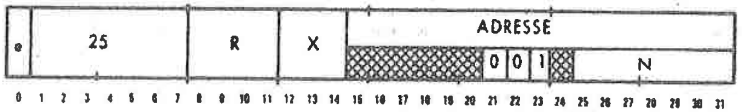


Si le nombre de positions de décalages N est positif, le contenu du registre R est décalé de N positions vers la gauche, des 0 étant placés dans les positions de c. b. rendues vacantes à droite. (Les c. b. décalés au-delà du c. b. 0 de R sont perdus).

Si N est négatif, le contenu du registre R est décalé de + N positions vers la droite, des 0 occupant la place des positions de c. b. rendues vacantes sur la gauche. (Les c. b. ayant dépassé le c. b. 31 de R sont perdus).

Sont modifiés : (R), CC1, CC2.

Décalage logique, double registre

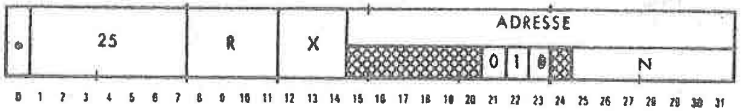


Si le nombre de positions de décalage N est positif, le contenu des registres R et Ru1 est décalé à gauche de N positions, des 0 étant placés dans les positions de c. b. rendues vacantes à droite.

Les c. b. débordant le c. b. 0 du registre Ru1 sont replacés dans le c. b. 31 du registre R. (Les c. b. débordant le c. b. 0 de R sont perdus). Si N est négatif, le contenu des registres R et Ru1 est décalé vers la droite de - N places, des 0 étant placés dans les c. b. rendus vacants à gauche. Les c. b. débordant le c. b. 31 du registre R sont recopiés dans le c. b. 0 du registre Ru1. (Les c. b. débordant le c. b. 31 du registre Ru1 sont perdus).

Sont modifiés : (R), (Ru1), CC1, CC2.

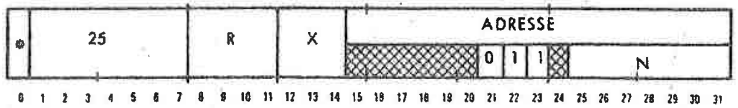
Décalage circulaire, registre unique



Si le nombre de positions de décalage N est positif, le contenu du registre R est décalé de N positions à gauche. Les c. b. débordant le c. b. 0 sont replacés dans le c. b. 31 (aucune c. b. n'est perdu). Si N est négatif, le contenu du registre R est décalé de - N positions vers la droite. Les c. b. débordant le c. b. 31 sont recopiés dans le c. b. 0 (aucun c. b. n'est perdu).

Sont modifiés : (R), CC1, CC2.

Décalage circulaire, double registre

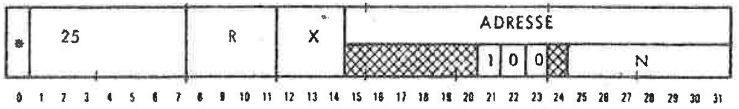


Si le nombre de positions de décalages N est positif, le contenu des registres R et Ru1 est décalé à gauche de N positions. Les c. b. débordant le c. b. 0 du registre R sont recopiés dans le c. b. 31 du registre Ru1. (Aucun c. b. n'est perdu).

Si N est négatif, le contenu des registres R et Ru1 est décalé à droite de - N positions. Les c.b. débordant le c. b. 31 du registre Ru1 sont recopiés dans le c. b. 0 du registre R. (Aucun c. b . n'est perdu).

Sont modifiés : (R), (Ru1), CC1, CC2.

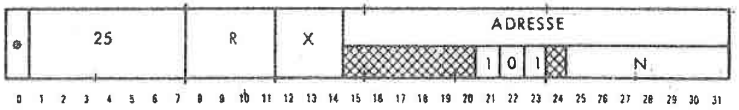
Décalage arithmétique, registre unique



Si le nombre de positions de décalage N est positif, le contenu du registre R est décalé à gauche de N positions, des 0 étant placés dans les c. b. rendus vacants à droite. (Les c. b. débordant du c. b. 0 de R sont perdus). Si N est négatif, le contenu du registre R est décalé à droite de - N positions, le contenu de c. b. 0 est recopié dans les c. b. rendus vacants à gauche. (Les c. b. débordant le c. b. 31 de R sont perdus).

Sont modifiés : (R), CC1, CC2.

Décalage arithmétique, double registre

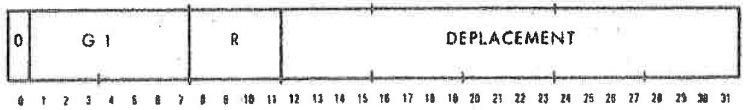


Si le nombre de positions de décalage N est positif, le contenu des registres R et Ru1 est décalé à gauche de N positions, des 0 étant placés dans les positions de c. b. rendues vacantes à droite. Les c. b. débordant le c. b. 0 du registre Ru1 sont recopiés dans le c. b. 31 du registre R. (Les c. b. débordant le c. b. 0 du registre R sont perdus). Si N est négatif, le contenu du c. b. 0 du registre R étant recopié dans les c. b. rendus vacants à gauche. Les c. b. débordant le c.b. 31 du registre R sont recopiés dans le c. b. 0 du registre Ru1. (Les c. b. débordant le c. b. 31 du registre Ru1 sont perdus).

Sont modifiés : (R), (Ru1), CC1, CC2.

MBS MOVE BYTE STRING

Déplacer chaîne d'octets
(adressage immédiat par octet, interruptible)



Cette instruction copie le contenu de la chaîne d'octets de l'émetteur (de gauche à droite) dans la chaîne d'octets du récepteur.
Le précédent contenu de la chaîne d'octets du récepteur est détruit mais le contenu de la chaîne d'octets de l'émetteur n'est pas modifié (sauf si la zone réceptrice recouvre la zone émettrice).

Sont modifiés : (DBS), (R), (Ru1)

(SBS) ⟶ DBS

On ne peut utiliser l'adressage indirect avec cette instruction, sinon elle est abandonnée, au moment de son décodage et un déroutement se produit vers l'emplacement X'40'. Dans ce cas, le contenu des registres R et Ru1 de même que celui de la chaîne d'octets du récepteur restent inchangés.

Il faut noter la manière dont cette instruction opère sur les mots et sur les octets pour être en mesure de prévoir le résultat final en cas de recouvrement des chaînes d'octets de l'émetteur et du récepteur. Par exemple, considérons le cas suivant où huit octets doivent être transférés et où la zone du récepteur recouvre la premier mot de la zone de l'émetteur :

Après le transfert du premier, puis du deuxième mot, les chaînes d'octets se présenteront de la façon suivante :

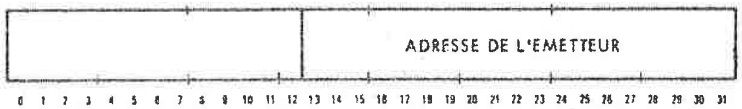
Si les chaînes d'octets de l'émetteur et du récepteur se chevauchent, il est préférable que la distance entre le premier octet de l'émetteur et le premier octet du récepteur ne soit pas inférieure à huit octets.

L'instruction MBS pourra être exécutée plus rapidement si les chaînes d'octets de l'émetteur et du récepteur commencent toutes les deux par le même octet à l'intérieur de leur mot respectif.

Ceci permet en effet d'effectuer le transfert (à l'exception peut-être du premier ou du dernier octet) par mots entiers.

•Cas I : R pair et différent de zéro (Ru1 = R+1)

Contenu du registre R

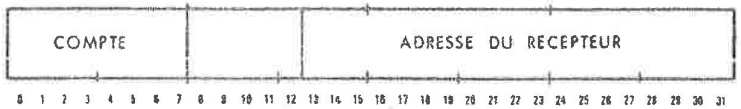


Contenu du registre R+1

L'émetteur commence à l'adresse obtenue en ajoutant le Déplacement (contenu dans l'instruction) à l'Adresse de l'émetteur (contenue dans le registre R). Le récepteur commence à l'Adresse du récepteur (contenu dans le registre R+1). Les deux chaînes comprennent C octets.
Lorsque l'instruction est terminée, les adresses de l'émetteur et du récepteur sont toutes deux augmentées de C et le Compte est égal à zéro.

•Cas II : R impair (Ru1 = R)

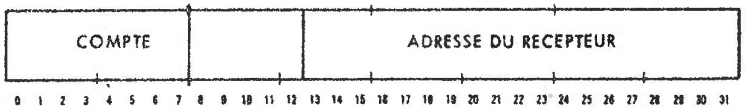
Contenu du registre R



L'émetteur commence à l'adresse obtenue en ajoutant le Déplacement (contenu dans l'instruction) à l'adresse du récepteur (contenue dans le registre R). Le récepteur commence à l'adresse du récepteur (contenue dans le registre R). Les deux chaînes comprennent C octets. Lorsque l'instruction est terminée, l'adresse du récepteur est augmentée de C et le compte est égal à 0.

•Cas III : R égal à zéro (Ru1 = 1)

Contenu du registre 1



L'émetteur ne comprend qu'un seul octet dont l'adresse est donnée par le Déplacement (contenu dans l'instruction).

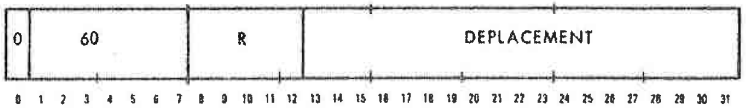
Le récepteur commence à l'Adresse du récepteur (contenue dans le registre 1); il comprend C octets.

Dans ce cas, l'octet constituant l'émetteur est reproduit sur toute la longueur de la zone du récepteur.

Lorsque l'instruction est terminée, l'adresse du récepteur est augmentée de C et le Compte est égal à zéro.

CBS COMPARE BYTE STRING

Comparer chaîne d'octets
(adressage immédiat par octet, interruptible)



L'instruction CBS compare, en valeur absolue, l'émetteur avec le récepteur. La comparaison est effectuée octet par octet en commençant par le premier octet des deux chaînes. Elle se poursuit jusqu'à ce que le nombre d'octets spécifié ait été comparé ou jusqu'à la rencontre d'une inégalité.

Lorsque cette instruction est terminée, CC3 et CC4 sont positionnés de manière à indiquer le résultat de la comparaison. Si l'instruction se termine par suite d'une inégalité, le Compte dans le registre Ru1 est égal au nombre d'octets restant à comparer PLUS UN; l'adresse de l'émetteur dans le registre R et l'adresse du récepteur dans le registre Ru1 indiquent alors la position des octets inégaux.

Sont modifiés : (R), (Ru1), CC3, CC4.
(SBS) :: (DBS)

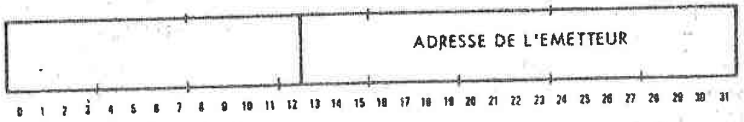
Positionnement du code de condition :

1	2	3	4	Résultat de CBS
-	-	0	0	émetteur égal au récepteur
-	-	0	1	émetteur intérieur au recepteur
-	-	1	0	émetteur supérieur au récepteur

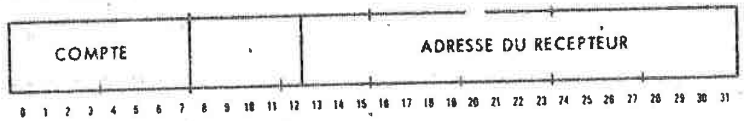
On ne peut utiliser l'adressage indirect avec cette instruction, sinon elle est abandonnée, au moment de son décodage, et un déroutement se produit vers l'emplacement X'40'. Dans ce cas, les contenus des registres R et Ru1, de CC3 et CC4 ne sont pas modifiés.

•Cas I : R pair et différent de zéro (Ru1 = R+1)

Contenu du registre R



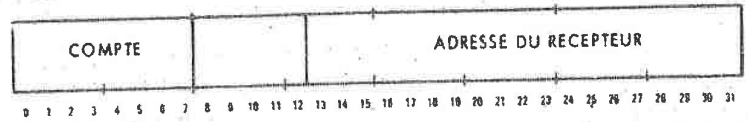
Contenu du registre R+1



L'émetteur commence à l'adresse obtenue en ajoutant le Déplacement (contenu dans l'instruction) à l'Adresse de l'émetteur (contenue dans le registre R). Le récepteur commence à l'adresse du récepteur(contenue dans le registre R+1). La longueur des deux chaînes est égale à C.

•Cas II : R impair (Ru1 = R)

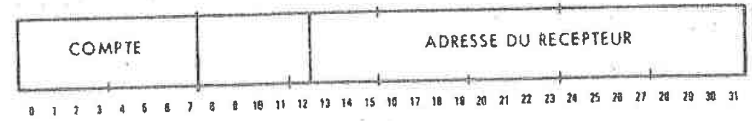
Contenu du registre R



L'émetteur commence à l'adresse obtenue en ajoutant le Déplacement (contenu dans l'instruction) à l'Adresse du récepteur (contenue dans le registre R). Le récepteur commence à l'Adresse du récep teur (contenue dans le registre R). La longueur des deux chaînes d'octets est égale à C.

•Cas III : R égal à zéro (Ru1 = 1)

Contenu du registre 1



L'émetteur ne comprend qu'un seul octet dont l'adresse est donnée par le Déplacement (contenu dans l'instruction). Le récepteur commence à l'Adresse du récepteur (contenue dans le registre 1); il comprend C octets. Dans ce cas, l'octet constituant l'émetteur est successivement comparé à tous les octets du récepteur.

CHAPITRE IV

UTILISATION D'ILM70

Ce chapitre présente l'utilisation qui a été faite des machines ILM70 en Travaux Pratiques du C2 d'Informatique Générale à Nancy. L'étude d'ILM70 sert d'application à la première partie du cours : présentation des ordinateurs et des techniques de programmation. Chaque notion exposée dans le cours est illustrée par l'utilisation d'une particularité d'ILM70 en travaux pratiques. Les éléves étudient simultanément la programmation en langage machine et l'écriture des programmes en SYMBOL (langage symbolique du 10070). Les machines ILM70 n'acceptant que des programmes numériques ils doivent donc faire une traduction manuelle de chaque programme.

En effet, il nous semble indispensable d'enseigner la programmation en langage machine, mais il faut bien sûr tenir compte de la commodité des langages d'assemblage. Nous insistons donc pour que les étudiants pratiquent les deux langages (machine symbolique) dés le début, pour 4 raisons :

1) Ils touchent de plus prés la réalité : seuls les programmes machine sont exécutables.

2) Ils deviennent vite très familiers du rôle exact de chaque directive d'un assembleur.

3) Ils comprennent rapidement, pour l'avoir fait eux-mêmes, le travail de l'assembleur. L'écriture en travaux pratiques d'un assembleur pour ILM70-3 se fait alors sans difficulté.

4) Pour avoir eu à résoudre des problèmes voisins sur des cas particuliers, ils suivent plus aisément les cours sur les problèmes de chargement et d'édition de liens, ainsi que sur les problèmes de gestion de la mémoire dans un compilateur.

Les étudiants abordent cette étude après une semaine d'étude d'un langage évolué (Fortran IV). Ils savent donc ce qu'est un organigramme, et connaissent très superficiellement quelques techniques de programmation : branchements, branchements conditionnels, boucles, rôle des entrées-sorties.

IV.1 Initiation à la programmation : Utilisation d'ILM70-1

Les problèmes posés par l'écriture et la mise en œuvre de programmes en ILM70 peuvent être abordés après la présentation sommaire de l'organisation d'une machine et de son fonctionnement. On précise qu'ILM70-1 est une machine fictive, légèrement différente du CII 10070 et que sur une vraie machine, on aurait des problèmes d'initialisation. On mentionne le programme de simulation et on explique qu'il comporte un chargeur très simple (on peut faire l'organigramme de ce chargeur) qui implante les programmes à l'adresse 10_{16}.

On précise également que les diagnostics d'erreurs sont faits par le programme de simulation. Dans un premier temps on n'utilisera qu'un sous-ensemble d'ILM70-1. En particulier, on se limitera aux instructions arithmétiques, de transfert et d'arrêt, sans adressage indirect ni indexation. On utilisera les "post-mortem dump" pour vérifier les résultats.

IV.1.1 Ecriture d'une instruction

Lorsqu'on n'utilise ni adressage indirect, ni indexation l'écriture d'une instruction ILM70-1 est aisée : les 2 premiers chiffres hexadécimaux représentent le code, le 3^{è} chiffre indique le registre concerné, les 2 chiffres suivants sont des 0, les 3 derniers chiffres sont l'adresse.

Après quelques exercices sur le calcul binaire et héxadécimal on rappelle la présentation d'une instruction en faisant rechercher le code, le registre et l'adresse dans une instruction donnée. Puis on écrit une instruction à partir de ces différents éléments. On peut passer rapidement à l'écriture de petits programmes.

IV.1.2 Ecriture des premiers programmes

Il s'agit en général de calcul d'expressions arithmétiques simples. Pour s'assurer de la bonne compréhension de cette notion de programme (description d'un travail à faire accomplir) et du fonctionnement de la machine pendant l'exécution, on fait "jouer les étudiants à la machine" : chacun d'eux étant, soit un registre général, soit le registre instruction, soit la fonction de lecture en mémoire... Ils déterminent ainsi les états successifs de la machine au fil d'une exécution, c'est-à-dire

les contenus des différents registres et de la mémoire. Ce "jeu" n'est effectif que lors des premières séances, mais il est toujours demandé de prévoir complétement les effets d'un programme en ayant suivi son déroulement. Le fonctionnement en pas à pas d'ILM70-1 permet de le vérifier. Le dump final montre que l'on trouve bien dans la mémoire et les registres les valeurs prévues. On complique ensuite les programmes en introduisant des branchements et des appels aux sous-programmes de lecture et d'écriture.

IV.1.3 Adressage indirect - Indexation

On a vu au chapitre I que la nécessité de ces techniques d'adressage apparait rapidement. Des exercices simples du type somme de n nombres rangés consécutivement en mémoire ou produit scalaire de 2 vecteurs permettent de les utiliser. Il est possible d'apprécier le gain de rapidité et et l'économie dans l'utilisation de la mémoire et des registres. Ces aspects sont d'ailleurs évoqués chaque fois qu'une nouvelle manière de programmer est exposée.

IV. 2 Apprentissage des techniques de programmation : utilisation d'ILM70-2

Alors qu'ILM70-1 permet l'introduction des notions de base, ILM70-2 est destinée à l'apprentissage des techniques de programmation, techniques qui resteront applicables jusqu'à la fin de l'enseignement puisque toutes les instructions d'ILM70-2 sont des instructions du 10070. On insiste en particulier sur l'écriture optimale des boucles, tests et sur l'écriture des sous-programmes. On peut, en Travaux Pratiques commencer à dégager une méthode pour passer manuellement du langage symbolique au langage machine ; on prépare ainsi la présentation faite en cours des méthodes d'assemblage.
On peut également mettre en évidence la nécessité de chargeurs translatables en faisant utiliser un même sous-programme par des programmes principaux différents. Les insuffisances du chargeur du programme de simulation apparaissent : on a besoin de lire autre chose que des nombres décimaux et pour faire des conversions il faudrait manipuler des caractères.

IV. 3 Utilisation d'ILM70-3 : Problèmes de chargement, d'assemblage, d'édition de lien

Après l'assimilation des notions nouvelles qui sont particulières à cette machine, on amène l'auditoire à concevoir pour ILM70-3 un certain nombre de programmes de service. Il serait long et inutile de les faire écrire tous, et pour certains d'entre eux, on se bornera à constater leur utilité et à en faire simplement l'analyse.

Dans un premier temps, on se limitera aux instructions de manipulation d'octets d'ILM70 et aux instructions d'entrée - sortie . Ces nouvelles instructions sont suffisantes pour écrire le chargeur.

IV. 3.1 Exemple d'énoncé de chargeur :

"Ecrire en langage machine ILM70-3 un programme qui lit et implante en Mémoire Centrale des programmes, des sous-programmes ou des données.

On trouvera sur chaque carte lue un nombre héxadécimal de 8 chiffres commençant à la colonne 1. Les 2 premiers nombres lus sont 2 adresses : la première est l'adresse d'implantation initialement prévue. La 2è est l'adresse nouvelle d'implantation. On trouve ensuite les nombres à implanter. Ils sont suivis d'un des 3 chiffres suivants précédé d'un blanc :

0 indique que le nombre lu n'est pas affecté par le changement d'implantation

1 indique qu'il s'agit d'une adresse de mot ou d'une instruction contenant une adresse de mot

2 indique qu'il s'agit d'une adresse d'octet.

Exemple : implantation à l'adresse X '100' d'un sous-programme écrit à partir de l'adresse X'37'

00000037	ancienne adresse	
00000100	nouvelle adresse	
35300052␣1	début du SSP : sauvegarde des	
35400053␣1	registres 3 et 4	
B230000A␣0	LW, 3 * 10	} inchangés
B2300003␣0	LW, 3 * 3	}
etc...		

Les différents programmes et sous-programmes à charger seront séparés par une carte comportant une ± en 1ère colonne. La fin des chargements est repérée par une carte comportant une ± dans les colonnes 1 et 2. On lit alors sur carte l'adresse de la première instruction à exécuter et on effectue un branchement à cette adresse.

Le traitement de la i^{è} instruction lue se fera pendant la lecture de la i+1^{è}. L'utilisation de ce chargeur permet d'aborder l'écriture de programmes importants, comme un assembleur, qui réutilisent des sous-programmes précédemment écrits, et qui peuvent être mis au point par morceaux. L'écriture d'un assembleur nécessite l'utilisation des instructions de mouvement et de comparaison de chaine d'octets d'ILM70-3.

IV. 3. 2 Définition du langage symbolique ILM70-3

Il s'agit d'une version extrêmement simplifiée de Symbol. Ce langage n'est d'ailleurs pas destiné à être utilisé effectivement, car dès la fin de l'écriture de l'assembleur, l'utilisation d'ILM70 est abandonné et les programmes de Travaux Pratiques sont écrits en Symbol.

Format d'une instruction symbolique sur carte :

```
TRUC␣␣␣ LW,5␣ ␣± A,3␣ ␣COMMENTAIRE
```

zone étiquette — zone commande — adresse — commentaire

La carte est divisée en 4 zones. Chaque zone est séparée de la suivante par un ou plusieurs blancs. Une zone ne contient aucun blanc.

- La zone étiquette commence en colonne 1 et peut être vide.
- La deuxième zone contient un code sur au maximum 4 caractères et éventuellement une "," suivie d'un nombre décimal compris entre 0 et 15.
- La troisième zone contient une adresse ou une constante qui peuvent être
  a) une chaine de 1 à 8 caractères (de même qu'une étiquette) ;
  b) une constante décimale de 1 à 8 chiffres éventuellement précédée d'un signe + ou - ;
  c) une constante héxadécimale de 1 à 8 chiffres précédée de X' suivie de ' ; exemple X 'FO'
  d) une constante de type caractère entre guillemets ; exemple : 'A' ou 'ABCD'
  e) une chaine de 1 à 8 caractères en opérande de BA ; exemple : BA (X) ;

Cette zone peut être précédée d'une ± et suivie d'une virgule et d'un chiffre décimal compris entre 1 et 7.
- La 4^{è} zone est ignorée.

On peut trouver, dans la 2^{è} zone, en plus des mnémoniques des instructions ILM70 une des directives suivantes :

RES, DATA, END

RES doit être suivi d'un entier décimal n. Cette directive provoque la réservation de n mots mémoires.

DATA peut être suivi de n'importe quelle adresse ou constante. Cette directive provoque la réservation d'un mot et son initialisation à la valeur indiquée.

END doit être suivi de l'étiquette de la première instruction à exécuter. Cette directive indique la fin du programme à assembler.

IV. 3. 3 Ecriture de l'assembleur

L'écriture de l'assembleur peut être préparée très tôt. On peut déjà, lors de l'utilisation d'ILM70-2 faire écrire des sous-programmes de recherche en table qui lui seront joints, pour la table des codes opérations et pour la table des symboles. Les codes opérations comportant au maximum 4 caractères seront rangés dans un mot, cadrés à gauche et éventuellement complétés par des blancs. On les classera par ordre alphabétique ce qui permet d'effectuer une recherche dichotomique. Cette recherche ne nécessitant pas la manipulation de caractères, on pourra la faire rédiger avant d'aborder l'étude d'ILM70-3.

Exemple d'énoncé : Recherche en table

"On a en mémoire un tableau TAB de 27 lignes et 2 colonnes, rangé ligne par ligne. Les 27 valeurs contenues dans la 1ère colonne sont rangées par ordre croissant : tab (i-1, 1) < tab (i, 1)
et sont toutes différentes.

Ecrire en langage machine ILM70-2 un sous-programme dont la séquence d'appel sera

```
BAL, 15    RECH
DATA       AD
```

dont le résultat sera la valeur (tab (i, 2) si (AD) = tab (i, 1), 0 si (AD) n'existe pas dans la 1ère colonne de TAB. Ce résultat sera mis dans le registre 14.

Ecrire un programme permettant de tester ce sous-programme."

Méthode de recherche : par dichotomies successives"

La recherche en table des symboles se fera de manière séquentielle cette table étant souvent modifiée. Les symboles comportant 8 caractères occuperont 2 mots qui seront suivis d'un mot destiné au rangement de l'adresse associée au symbole. On pourra, en utilisant ILM70-2 écrire un sous-programme auquel on donne un symbole et qui donne pour résultat, si le symbole n'existe pas dans la table, 0, sinon le contenu du mot qui lui est associé.

Avant d'aborder l'écriture de l'assembleur proprement dit on peut écrire un programme qui prépare l'instruction symbolique à un traitement ultérieur.

Exemple d'énoncé :

"Ecrire un programme qui lit une instruction symbolique sur carte et qui remplit la suite de mots suivants :

1er et 2ème mot : le symbole de la zone étiquette cadré à gauche complété par des blancs, ou des zéros si la zone est vide ;

3ème mot : le code de la zone commande cadre à gauche complété par des blancs

4ème mot : la constante de la zone code ou 0 si elle est absente

5ème mot : 0 ou 1 suivant qu'il y a ou non une astérisque en zone adresse

6ème mot :
- 1 si l'adresse est un symbole
- 2 si c'est une constante décimale positive
- 3 si c'est une constante décimale négative
- 4 si c'est une constante héxadécimale
- 5 si c'est une chaine de caractères
- 6 si c'est un symbole précédé de BA ;

7ème et 8ème mot : Dans les cas 1 et 6 la chaine de caractères définissant symbole, cadrée à gauche et complétée par des blancs ; Dans les cas 2 et 3, le nombre décimal duquel on a supprimé le signe, cadré à gauche et complété par des blancs ; dans les 4 et 5, la chaine de caractères qui était entre guillemets, cadrée à gauche, complétée par des blancs ;

9ème mot : le nombre compris entre 1 et 7, figurant en zone adresse.
0 s'il n'y en a pas."

exemple :

	TRUC	LW, 5		* A, 3				
TRUC	␣␣␣␣	LW␣␣	5		1	1	A␣␣␣	␣␣␣␣	3

		LI, 9		X 'F0'				
0	0	LI␣␣	9	0	4	F0␣␣	␣␣␣␣	0

Cet assembleur est à un passage : ILM70-3 ne permet pas de stocker le programme source et, le 10070 étant exploité par train, il n'est pas non plus possible de lire 2 fois les cartes source. Les parties difficiles à programmer sont celles relatives aux chainages que nécessite le traitement des références en avant. On pourra dans un premier temps ne pas écrire ces traitements mais leur prévoir une implantation. On fait écrire le reste de l'assembleur comme s'ils étaient réalisés. Il est alors possible de tester l'assembleur dans le cas de programmes sans référence en avant ; on lui ajoute ensuite les traitements manquants. Le résultat de l'assemblage est un programme en mémoire à une adresse fixe à la place du chargeur qui a été utilisé pour implanter les divers éléments de l'assembleur en mémoire. Après l'écriture du chargeur et de l'assembleur, on peut demander à l'auditoire de critiquer ces deux programmes et de proposer des améliorations. La première critique concerne certainement le chargeur car ce programme a été beaucoup plus utilisé que l'assembleur. Il faut en effet prévoir l'implantation de tous les sous-programmes avant d'écrire le programme principal ce qui est peu commode et risque de faire perdre de la place en mémoire. On cherche qu'elles sont les données nécessaires à un programme qui prévoirait l'implantation des sous-programmes et compléterait le programme principal. La notion de définition externe et de référence externe est alors introduite très naturellement. Une autre critique, relative à l'assembleur, et que chaque exécution nécessite un assemblage. On pense naturellement à un perforateur de carte, et on remarque que si on veut obtenir des programmes implantables par le chargeur, il faut joindre à chaque instruction sa marque de translation, et que si on assemble séparément un programme principal et des sous-programmes, on retrouve le même genre de problèmes que ceux posés par l'amélioration du chargeur. Le résultat de l'assemblage doit donc comporter les données nécessaires à ce chargeur amélioré : on peut alors parler des directives REF et DEF de Symbol. L'auditoire se familiarise ainsi avec les notions de base

relatives à l'édition de liens, et pourra aborder, sans trop de difficultés, l'étude des programmes de service et des systèmes associés à une machine. Il possède également une connaissance solide des mécanismes d'assemblage et d'un sous-ensemble de Symbol.

L'utilisation effective des machines ILM70 a démontré clairement l'intérêt : l'initiation est facilitée, la progression dans l'enseignement est accelérée, et les étudiants abordent l'étude du 10070 sans difficultés.

CHAPITRE V

SIMULATION D'ILM70

INTRODUCTION

ILM70-1, ILM70-2 et ILM70-3 ont été simulées sur CII 10070. Les 3 programmes de simulation correspondants ont été nommés NIV1, NIV2, NIV3. Ces programmes sont très proches quant à leur organisation logique, et ne diffèrent que par des points de détails. Par exemple NIV1 et NIV2 ne diffèrent que par 2 points : la liste des codes opérations acceptés et la simulation des instructions de branchements.

Ces programmes seront donc présentés conjointement dans le paragraphe 1.

Par contre le programme NIV3, qui permet de simuler des entrées-sorties simultanées a une organisation plus compliquée. Il sera présenté dans le paragraphe V. 2.

D'autre part, on a utilisé au maximum, dans l'écriture de ces programmes, le fait que les instructions ILM70 étaient, pour la plupart des instructions du 10070. Ces programmes ne sont donc pas vraiment des programmes de simulation, mais plutôt des programmes de commande et de contrôle, qui permettent à des programmeurs débutants d'écrire en langage machine sans perturber le passage des autres programmes, de disposer de diagnostics d'erreur clairs, de "post mortem dump" etc ... .

L'écriture de ces programmes a été facilitée (donc influencée) par l'existence d'un certain nombres d'instructions 10070, particulièrement bien adaptées à ce genre de problèmes.

Il s'agit de l'instruction EXU qui provoque l'exécution de l'instruction contenue dans le mot d'adresse effective, et de l'instruction ANLZ qui calcule l'adresse effective de l'instruction contenue dans le mot d'adresse effective.

V. 1 Organisation Générale de NIV1 et NIV2

Ces programmes sont formés de 3 parties qui sont exécutées séquentiellement :

(CHARG) chargement du programme ILM70

(INIT) initialisation d'ILM70

(INTER) interpréte du programme ILM70.

V. 1. 1. Chargement du programme ILM70

La Mémoire Centrale de ILM70 est contenue dans les zones suivantes de la mémoire du 10070 :

a) les registres généraux 0 à 15 pour les 16 premiers mots

b) les mots MEM + 16 à MEM + dim pour les mots suivants. Dans la version actuelle dim représente la constante 4095. Pour modifier la dimension de la mémoire il suffit de changer la constante dim.

L'opération de chargement se décompose de la manière suivante :

on lit des cartes jusqu'à la rencontre d'une carte !EØD. Lorsqu'on rencontre une telle carte, on passe à la partie (INIT) (initialisations).

Pour chaque carte lue, on effectue les opérations suivantes :

(CHARG1) Impression des 80 caractères lus

(CHARG2) Conversion des 8 premiers caractères, considérés comme 8 chiffres hexadécimaux, en un nombre binaire. Si un des caractères est différent de 0, 1 ..., 9, A, ..., F il y a impression du diagnostic

'ERREUR A LA LECTURE'

et arrêt.

(CHARG3) Rangement du résultat de la conversion à l'adresse MEM + 15 + i, où i est le numéro de la carte lue. Les valeurs lues sont donc rangées séquentiellement à partir de l'adresse MEM+16.

V. 1. 3 Initialisations

Cette partie du programme comprend essentiellement les ordres initiaux permettant la mise en marche d'ILM70. Ici, il s'agira

uniquement de mettre la valeur 16 dans le compteur ordinal et d'effectuer un branchement au début de l'interprête (partie (INTER) du programme). De plus, pour obtenir une plus grande clarté des "Dump Mémoire" futurs, les registres généraux sont mis à zéro.

V. 1. 3 Interprête

Cette partie du programme simule le fonctionnement de ILM70. Chaque élément de ILM70 doit donc être représenté dans la mémoire du 10070 et sera modifiée par l'interprête.

V. 1. 3. 1 Représentation dans le programme de simulation des principaux éléments de ILM70.

1) Les 16 registres généraux de ILM70 sont les registres généraux du 10070.

Cette représentation permet, lorsque l'instruction ILM70 à exécuter est une instruction du 10070, d'en demander simplement l'exécution après avoir éventuellement ajouté MEM à la zone adresse (voir V. 1. 1). Dans le cas de NIV2, pour positionner le code de condition ILM70, il suffit de ranger le code de condition du 10070 dans le code de condition ILM70-2. Cependant, cette représentation nécessite de sauvegarder et de restaurer les registres du 10070 et le code de condition , après toute utilisation de ces registres par NIV1 ou NIV2.

2) Les registres internes de ILM70 sont représentés par les mots mémoires suivants :

CØR pour le compteur ordinal

RI pour le registre instruction

CC pour le code de condition. Seul les 4 premiers bits sont utilisés.

3) La représentation de la Mémoire Centrale d'ILM70 est décrite en III. 1. 1.

V. 1. 3. 2 Description de l'interprête

Pour chaque instruction du programme ILM70 on effectue les opérations :

1) CINST : chargement dans le registre instruction du contenu du mot

d'adresse MEM + (CØR) si (CØR) ⩾ 16, (CØR) sinon.

2) PROG : Progression du compteur ordinal
3) TAI : Ttraitement de l'adressage indirect
4) TEST : Si (RI) = instruction d'arrêt, Arrêt
5) AIG : Aiguillage selon le code opération

Après AIG, on passe à l'un des 3 traitements suivants :

6) SEQU : Traitement d'une instruction ne provoquant pas de rupture de séquence
7) BRCHT : Traitement d'une instruction de branchement
8) TBAL : Traitement de l'instruction BAL

après ces traitements on exécute :

9) RET : Retour à CINST (passage à l'instruction suivante du programme d'ILM70).

(PROG) : Progression du compteur ordinal

On ajoute 1 au contenu de CØR. On teste ensuite, comme chaque fois que l'on modifie CØR, si la valeur obtenue est une adresse d'ILM70, c'est-à-dire un nombre compris entre 0 et dim (cf V.1.1). Si ce n'est pas le cas, il y a impression du diagnostic

'Adresse incorrecte'

et arrêt.
Remarquons qu'aucune instruction ILM70-3 n'étant "interruptible" il est logique de faire la progression à cet endroit.

(TAI) : Traitement de l'adressage indirect

Si (RI) ⩾ 0, il n'y a pas d'adressage indirect, et l'on va exécuter directement (TEST)
Sinon, on effectue les opérations suivantes :

(TAI.1) : Remettre à 0 le bit 0 de RI

(TAI.2) : Notons AD le contenu des 17 derniers bits de RI. On charge à la place de ces 17 derniers bits les 17 derniers bits du mot MEM+AD si AD > 15, du registre AD si AD ⩽ 15.

On obtient ainsi, dans RI, une instruction ILM70 équivalente à celle qui y était précédemment, mais sans adressage indirect.

(AIG) : Aiguillage selon le code opération

Cette partie du programme est différente pour NIV1 et NIV2. En effet, le jeu d'instructions, donc la table des codes opérations et leur nombre différent dans ces 2 programmes.

a) Représentation du jeu d'instructions autorisées :

Le nombre d'instructions autorisées est représenté par la constante nco. Lorsque l'on modifie le jeu d'instructions, il est donc nécessaire de modifier cette constante.

Dans la version actuelle on trouve :

dans NIV1	NCO = 10
dans NIV2	NCO = 18

La table des codes opérations autorisés, comprend donc nco éléments, et est implantée à partir de l'adresse TAB. Les codes opérations y sont rangés par ordre croissant.

Remarque : Le code opération de l'instruction d'arrêt n'apparait pas dans cette table, et n'est pas compté dans nco. Cette instruction est en effet traitée à part dans la partie (TEST)

b) Description de l'aiguillage

Si le code opération testé, n'est pas celui d'une instruction de branchement, on passe à l'exécution de (SEQ) traitement d'une instruction ne provoquant pas de rupture de séquence.

Dans NIV2, ces instructions peuvent être de 2 types : à adressage immédiat ou à adressage direct. On a donc prévu 2 entrées dans la partie (SEQ) du programme NIV2, correspondant à chacun de ces 2 cas.

Dans le cas d'instructions de branchement, on passe aux traitements spécifiques de ces instructions qui différent évidemment totalement dans NIV1 et NIV2. Dans le $i^{è}$ mot à partir de TAB1, on trouve une instruction de branchement qui renvoie au traitement correspondant au $i^{è}$ code opération de TAB.

(SEQ) : Traitement des instructions ne provoquant pas de rupture de séquence

Il s'agit d'instructions de transfert, de comparaison ou arithmétiques. Elles sont identiques aux instructions correspondantes du 10070.

Leur simulation est donc superflue et on en demandera simplement l'exécution par l'intermédiaire de l'instruction EXU (voir annexe en fin de ce chapitre). Il faut cependant préparer cette exécution : on effectuera donc les opérations suivantes,

(SEQU. 1) : Translation de la zone adresse (c.a. d. addition de MEM au au contenu de RI) dans le cas où l'adresse effective est strictement supérieure à 15.

(SEQU. 2) : Restauration de l'environnement
ILM70 qui a été éventuellement modifié par NIV1 et NIV2 (il s'agit uniquement des registres généraux utilisés dans NIV1 et NIV2, et du code de condition dans NIV2).

(SEQU. 3) : Commande d'exécution de l'instruction contenue dans RI

(SEQU. 4) : Sauvegarde de l'environnement ILM70 en prévision de l'exécution de l'instruction suivante.

Dans le cas des instructions à adressage immédiat de ILM70-2, il ne faut pas exécuter (SEQU. 1) ; on a donc prévu, dans NIV2, une deuxième entrée dans ce traitement qui correspond à (SEQU 2)

On passe ensuite à l'exécution de (RET) .

(BRCHT) : Traitement des instructions de branchement
Ce traitement diffère dans NIV1 et NIV2.

a) Traitement des instructions de branchement dans NIV1.

Ces instructions peuvent être BRI, BRP, BRZ. A chacune d'entre elles correspond un traitement particulier :

(BRI) : Traitement du branchement inconditionnel
On calcule à l'aide de l'instruction ANLZ l'adresse effective de cette instruction . On la range dans CØR après avoir testé s'il s'agit d'une adresse ILM70 (cf desciption de (PROG)). On passe ensuite à l'exécution de (RET) .

(BRP) : Traitement de BRP
Ce traitement se décompose de la manière suivante :

(BRP. 1) Identification du registre à tester.

(BRP. 2) Restauration des registres ILM70-1
BRP ne modifiant pas les registres, la sauvegarde sera inutile.

(BRP. 3) Test du contenu du registre. Si il est strictement positif, on va exécuter (BRI), c'est-à-dire un branchement inconditionnel. Sinon on passe directement à l'exécution de (RET) .

(BRZ) : Traitement de BRZ.

Ce traitement est identique à celui de BRP, mais dans la 3ème partie, on va exécuter (BRI) si le contenu du registre est nul.

Remarque : (BRP. 2) est différent de (SEQU. 2) : les traitements qui les précédent n'utilisent pas le même nombre de registres.

Traitement des instructions de branchement dans NIV2

On a conservé dans NIV2 la partie (BRI) (traitement du branchement inconditionnel) de NIV1, mais on ne s'y branche plus directement depuis (AIG) (aiguillage).

Le traitement des instructions de branchement de ILM70-2 s'effectue par la même séquence d'instructions, qu'il s'agisse de BCR, BCS, BIR et BDR. On utilise en effet le fait que ces instructions sont des instructions du 10070 : Si dans RI on a un "branchement si condition P", la simulation de ce branchement sera faite dans NIV2 de la manière suivante :

"branchement si condition P" au traitement du branchement inconditionnel, cette instruction étant construite par NIV2 à partir du contenu de RI puis exécutée.

On peut donc décrire ce traitement :

(BRCHT) : Traitement de BCR, BCS, BIR, BDR

(BRCHT. 1) Construction à partir des 12 premiers bits de RI de l'instruction conditionnelle

BCR
BCS
BIR
BDR } , R BRI BRI étiquetant le début de (BRI)

et on la range au début de (BRCHT. 3) .

(BRCHT. 2) Restauration de l'environnement ILM70.

(BRCHT. 3) Exécution de l'instruction construite en (BRCHT. 1) : si la condition est vérifiée on passe à l'exécution de (BRI).

(BRCHT. 4) Sauvegarde de l'environnement ILM70 et passage à l'exécution de (RET).

(BRI) Traitement d'un branchement inconditionnel.

Le 3ème traitement où l'on peut se brancher depuis (AIG) est le traitement de BAL :

(TBAL) a) Traitement de BAL dans NIV1

Ce traitement peut être représenté par la table de décision suivante :

(RI) = BAL, 15 LEC	V	F	F
(RI) = BAL, 15 ECR	F	V	F
(TBAL. 1)	1		
(TBAL. 2)		1	
(TBAL. 3)			1
(TBAL. 4)	2	2	
(RET)	3	3	

où

(TBAL. 1) représente : branchement au sous-programme de lecture.

(TBAL. 2) Branchement au sous-programme d'écriture.

(TBAL. 3) Impression du diagnostic d'erreur 'Adresse incorrecte' et arrêt.

(TBAL. 4) Mettre dans le registre 15 d'ILM70 la valeur (CØR).

(TBAL) b) Traitement de BAL dans NIV2.

La description donnée pour NIV1 reste valable, seule la partie (TBAL. 3) est modifiée :

(TBAL. 3) Simulation de BAL

(TBAL. 31) Identification du registre designé par l'instruction (registre de liaison).

(TBAL. 32) Rangement de (CØR) dans ce registre.

(TBAL. 33) Branchement à l'exécution de (BRI) (traitement du branchement inconditionnel).

V. 1. 4 Quelques remarques sur l'écriture de NIV1 et NIV2

L'instruction EXU du 10070 peut avoir pour opérande effectif une instruction où l'adressage indirect est utilisé. Mais du fait que les adresses d'ILM70 doivent être translatées, il est nécessaire de remplacer toutes les instructions ILM70 comportant une indirection, par l'instruction équivalente où l'indirection a été effectuée, ce qui est fait en (TAI) (V. 1. 3. 2).

En effet, considérons l'instruction ILM70 suivante :

INST, R * AD

son adresse effective ILM70 est (AD), c. a. d. le contenu du mot ILM70 d'adresse AD,
et l'adresse correspondante du 10070 est :

(MEM + AD) + MEM.

Or, si on demande l'exécution de cette instruction par un EXU après l'avoir simplement translatée de MEM, l'adresse effective sera (MEM + AD), donc, incorrecte.

D'autre part, lorsque l'instruction EXU a pour opérande une instruction de branchement, le branchement a lieu. Si on avait utilisé EXU pour simuler les branchements de ILM70-2, NIV2 aurait donc perdu le controle de l'exécution du programme ILM70.

Les sous-programmes de lecture et d'écriture sont 2 sous-programmes internes à NIV1 et NIV2. Ils utilisent 15 comme registre de liaison, ce qui permet, comme on chargera ensuite ce registre avec (CØR), de ne pas le sauvegarder avant les appels.

V. 2 Description de NIV3

On trouve dans NIV3 le même découpage en 3 parties que dans NIV1 et NIV2 :

(CHARG) Chargement

(INIT) Initialisations

(INTER) Interprète.

La partie (CHARG) est inchangée par rapport à NIV1 et NIV2. On a ajouté, dans la partie (INIT) l'initialisation à zéro de 2 variables T et U qui sont des 'indicateurs de temps d'attente' dans les opérations d'entrées-sorties. Leur rôle est décrit en V 2.2. Dans la partie INTER , de nombreuses parties sont inchangées. Rappelons en l'organisation :

(INTER) Interpréte du programme ILM70-3

(CINST) Chargement dans le registre instruction RI du contenu de MEM + (CØR) si (CØR) ≥ 16, de (CØR) si (CØR) < 16.

(PROC) Progression du compteur ordinal.

(TAI) Traitement de l'adressage indirect.

(TEST) Si (RI) = instruction d'arrêt, Arrêt.

Ces 4 premières parties sont inchangées par rapport à NIV1 et NIV2.

(AIG) Aiguillage selon le code opération, vers l'un des 6 traitements suivants :

(SEQ) Traitement des instructions ne provoquant pas de rupture de séquence sauf SIØ, TIØ, MBS, CBS.

(BRCHT) Traitement de BCR, BCS, BIR, BDR.

(TBAL) Traitement de l'instruction BAL.

Ces 3 dernières parties sont inchangées par rapport à NIV2.

(ICØCT) Traitement des instructions sur chaines d'octets (MBS ou CBS).

(SSIØ) Simulation de SIØ

(STIØ) Simulation de TIØ.

Après chacune de ces 6 opérations on passe à l'exécution de

(GES) Gestion des entrées-sorties en attente et retour en CINST.

La description des traitements identiques à NIV1 et NIV2 ne sera pas reprise ici. On trouvera donc uniquement les descriptions de : (AIG) , (ICØCT), (SSIØ) , (GES) , (STIØ) .

V. 2. 1 Aiguillage :

La table des codes opérations autorisés comprend 26 éléments qui sont rangés par ordre croissant à partir de l'adresse TAB. La constante nco a donc la valeur 26. On trouvera dans TAB1 7 sortes d'instructions de branchement correspondant aux 6 traitements décrits ci-dessus, la partie SEQ possédant 2 entrées (cf. V. 1. 3. 1).

V. 2. 2 (ICØCT) Traitement des instructions sur chaines d'octets

Ces instructions sont des instructions du 10070, dont le déroulement est relativement complexe. Il est donc intéressant d'en demander l'exécution par l'intermédiaire d'une instruction EXU. Il faut cependant effectuer un certain nombre d'opérations auparavant, car l'opération à exécuter porte sur des zones de la mémoire d'ILM70. Cette mémoire est, on l'a vu en V. 1. 1, représentée par 2 zones de la mémoire du 10070 : les 16 registres du 10070 pour les 16 premiers mots (ou les 16 registres d'ILM70), les mots d'adresse Mem + 16 à Mem + dim du 10070 pour les mots suivants d'ILM70. Cette représentation pose un problème car on peut avoir pour opérande des instructions MBS et CBS des chaines de caractères commançant dans la 1ère zone et finissant dans la seconde. On résoud cette difficulté en plaçant les 16 registres ILM70 dans les mots Mem à Mem+15 du 10070 et en translatant de 4 x Mem les adresses des chaines d'octets opérandes avant de demander l'exécution.

Cependant, les registres ILM70 ne jouent pas seulement un rôle d'opérande éventuel dans ces instructions, le registre R designé par l'instruction (et éventuellement RU1) contient les renseignements sur l'opération a effectuer et sont mis à jour durant le déroulement de l'instruction après chaque opération élémentaire (transfert ou comparaison d'1 octet, transfert de 4 octets). A la fin de l'exécution de l'instruction ils ont été modifiés. Au moment de demander l'exécution de l'instruction, les registres R et RU1 du 10070 doivent

donc contenir les registres R et RU1 d'ILM70. A la fin de l'exécution de l'instruction, on replacera les registres d'ILM70 dans les registres du 10070 pour retrouver la représentation habituelle, mais il faudra conserver les contenus de R et RU1, puisque les modifications apportées à leurs contenus doivent apparaître dans ILM70.

Dans le cas où des registres ILM70 jouent le double rôle d'opérandes et de mots de commande dans une instruction MBS ou CBS, le programme de simulation provoque l'impression du diagnostic

' instruction incorrecte'

et l'arrêt. Ce cas n'est pas interdit sur 10070, mais nous avons préféré qu'il le soit sur ILM70 : le transfert ou la comparaison de chaines de caractères dont le contenu évolue pendant le déroulement de l'instruction n'a généralement pas grand sens, et la simulation d'ILM70-3 en aurait été considérablement alourdie.

Le traitement des instructions sur chaines d'octets se décompose en :

(ICØCT. 1) Rangement des 16 registres d'ILM70 dans les mots MEM à MEM+15.

(ICØCT. 2) Identification des registres désignés dans l'instruction : R, RU1.

(ICØCT. 3) Test de correction de l'instruction.

(ICØCT. 4) Translation du contenu de RU1, et si R est pair, du contenu de RI. Restaurer R et RU1.

(ICØCT. 5) Exécution de l'instruction contenue dans RI.

(ICØCT. 6) Soustraire du contenu de RU1 la quantité ajoutée en (ICØCT. 4) Ranger R et RU1 dans Mem+R et Mem+RU1. Charger dans les 16 registres les mots Mem à Mem+15.

(ICØCT. 7) Branchement à l'exécution de (GES) .

Test de correction de l'instruction.

(ICØCT. 3) : On vérifie que les zones R et RU1 et la zone émettrice, ou la zone réceptrice, ne se recouvrent pas.

ICØCT. 3. 1 Calcul en adresses d'octets, des 2 limites de la zone R, RU1, qui seront notée 'dreg' et 'freg'.

ICØCT. 3. 2 Test de correction par rapport à l'émetteur :

On calcule, en adresse d'octet, le début de l'émetteur. Si cette adresse est supérieure à freg, la zone R, RU1 et l'émetteur ne se recouvrent pas et on passe à ICØCT. 3. 4. Sinon, on calcule la fin de l'émetteur. Si cette adresse est inférieure à dreg les 2 zones ne se recouvrent pas et on passe à ICØCT. 3. 4 sinon, on passe à ICØCT. 3. 3.

ICØCT. 3. 3 Impression du diagnostic d'erreur, restauration des registres - Arrêt.

ICØCT. 3. 4 Test de correction par rapport au récepteur : le principe est le même que dans ICØCT. 3. 2.

V. 2. 3 Simulation des entrées-sorties en simultanéité d'ILM70-3

V. 2. 3. 1 Principe

NIV3 doit donner à l'utilisateur d'ILM70-3 la possibilité de gérer des entrées-sorties simultanément au traitement de l'unité centrale. Or, on a vu que les programmes NIV1, NIV2, NIV3 devaient permettre le passage de programmes ILM70 d'une manière commode, sans qu'il soit nécessaire de modifier le mode d'exploitation utilisé sur 10070. En effet, NIV3 travaille en mode esclave, sous BPM*. Cela signifie en particulier que les opérations d'entrées-sorties de NIV3 sont effectuées par l'intermédiaire de procédures du système, sous-contrôle de celui-ci. Dans le cas du lecteur de cartes et de l'imprimante, les opérations d'entrées-sorties sont faites en simultanéité avec le traitement de l'unité centrale, mais l'utilisateur n'a pas à gérer ces simultanéités. En particulier, dans le cas d'une lecture de carte, le résultat de la lecture est disponible instantanément au moment où on demande la lecture, car celle-ci à déjà été effectuée par le système.

Les entrées-sorties de NIV3 auront donc toujours lieu sans simultanéité, mais on donnera au programmeur ILM70-3 l'impression de cette simultanéité :

- le résultat d'une entrée-sortie lancée par une SIØ ne sera disponible qu'après un délai aléatoire qui représentera le temps d'exécution de l'entrée-sortie.

- lorsque NIV3 rencontre une instruction SIØ dans un programme ILM70, l'exécution de l'opération d'entrée-sortie demandée est différée du temps

* BPM : Batch Processing Monitor - Manuel d'utilisation.

d'interprétation d'un certain nombre d'instructions ILM70-3, ce nombre variant d'une opération à l'autre.

V. 2. 3. 2 Utilisation des indicateurs de temps d'attente dans NIV3

ILM70-3 possède 2 périphériques qui sont une imprimante et un lecteur de cartes.

A chacun de ces 2 périphériques, on a associé dans NIV3, 1 variable qui joue le rôle d'"indicateur de temps d'attente". Ces variables sont T pour le lecteur de cartes, U pour l'imprimante. Si ces variables ont des valeurs nulles, cela signifie qu'il n'y a pas d'entrées-sorties "en cours" pour ILM70-3. Si une de ces variables a une valeur non nulle il y a une entrée-sortie ILM70 "en cours" sur le périphérique associé, l'exécution de cette entrée-sortie a été différée par NIV3, et elle aura lieu après T ou U interprétations d'instructions ILM70. Chaque fois que l'on interprète une instruction ILM70, on décrémente de 1 les indicateurs de temps d'attente non nuls. Si cette décrémentation annule l'indicateur, on exécute l'entrée-sortie qui avait été différée par NIV3.

Ces indicateurs permettent également de tester si un périphérique est "occupé" au sens ILM70 ou non (c'est-à-dire qu'une entrée-sortie a été lancée mais n'est pas encore terminée). Ils sont initialisés à zéro dans la partie INIT de NIV3.

V. 2. 3. 3 Simulation de SIØ ((SSIØ))

Lors de l'exécution d'une instruction SIØ il peut se présenter 4 cas :

1) l'adresse d'entrée-sortie contenue dans l'instruction désigne le lecteur de cartes.
2) L'adresse d'entrée-sortie désigne l'imprimante.
3) L'adresse d'entrée-sortie n'a pas été reconnue : ou imprime le diagnostic : 'adresse E/S inconnue'.
4) Le départ d'entrée-sortie n'a pas réussi : on imprime le diagnostic : 'départ d'E/S non réussi'.

Ces 2 derniers cas provoquent également le positionnement du code de condition comme il est dit en III. 4. 5.

Le cas où l'adresse d'entrée-sortie contenue dans l'instruction désigne le lecteur de cartes est traité ainsi :

Si le contenu de T (indicateur de temps associé au lecteur de cartes) est nul, on affecte à T une valeur comprise entre 10 et 16, on range le contenu du registre 0 dans une mémoire de travail ; si le contenu de T n'est pas nul, il s'agit d'un départ non réussi et on passe au traitement de ce cas. Dans le cas d'une instruction SIØ relative à l'imprimante, le traitement est identique, seul le nom de l'indicateur de temps diffère.

V. 2. 3. 4 Simulation de TIØ ((STIØ))

Si l'adresse d'entrée-sortie contenue dans l'instruction TIØ est reconnue, on teste si l'indicateur de temps associé au périphérique désigné par cette adresse est nul. Si oui, on peut lancer une entrée-sortie, sinon on ne peut pas. On positionne le code condition, en fonction du résultat de ce test, comme il est dit en III. 4. 5.

V. 2. 3. 5 Gestion des entrées-sorties en attente ((GES))

On passe dans cette partie du programme après chaque interprétation d'une instruction d'ILM70-3.

On y effectue les tests et traitements suivants :

T = 0 ?	V	V	F	F
U = 0 ?	V	F	V	F
GES. 1			1	1
GES. 2		1		2
GES. 3	1	2	2	3

(GES. 1) Décrémentation de T de 1. Si le résultat est nul, 2 cas peuvent se présenter : si l'adresse de commande associée à la SIØ est celle du jeu standard de commandes (cartes) exécution de la lecture de cartes définie en III. 4. 6, sinon impression du diagnostic

'commande UE incorrecte'

et arrêt.

(GES. 2) Décrémentation de U de 1, si le résultat est nul, le traitement est identique à celui décrit en (GES. 1), mais cette fois pour l'imprimante.

(GES. 3) Retour en (CINST).

Remarque : Les retours sur conditions particulières lors de l'exécution des E/S d'ILM70-3 auront pour effet :

dans le cas d'une lecture, impression du diagnostic

'ERREUR DE LECTURE'

et abandon du programme.

Dans le cas d'une impression, impression du diagnostic

'IMPRESSION ABANDONNEE'

et poursuite du programme.

V. 2. 2. 5 Exemple de fonctionnement de NIV3

Prenons un exemple de programme ILM70-3 qui effectue des opérations d'entrées-sorties en simultanéité. Nous allons étudier ce qui se passe dans l'Unité Centrale et l'Unité d'Echange fictives d'ILM70, et, parallélement l'évolution dans NIV3 du compteur de temps d'attente et d'un certain nombres de variables.

Les instructions de ce programme sont écrites en symbolique ILM70-3 pour en faciliter la lecture.

```
TEST1   TIØ, 0      CRA03     test de l'état du lecteur de cartes
        BCS, 4      TEST1     Si le lecteur est occupé, on boucle
                              sur le test
        LI, 0       X'1000'   Dans le registre 0, on met l'adresse
                              (divisée par 2) du jeu standard de
                              commandes (cartes)
        SIØ, 0      CRA03     Lancement de la lecture
        BCS, 4      ERD       En cas de départ non réussi, on va en
                              ERD
```

instruction n° 1

|

instruction n° 10


```
TEST2   TIØ, 0      CRA03     Test de l'état du lecteur
        BCS, 4      TEST2     Si le lecteur est occupé, on boucle
                              sur le test.
```

instruction n° 13

ERD ~~~~~~~~~~~~~~~~

NIV3	Unité Centrale d'ILM70-3		Unité d'Echange d' d'ILM70-3
Dépouillement de l'instruction Branchement à (STIØ) : Adresse E/S = CRA03 ⇒ Test sur T T = 0 ⇒ CC := 0	TIØ, 0	CRA03 →	Reconnaissance du périphérique. Envoi à l'U. C. de son état : "libre"
	positionnement du code de condition à 0		
GES sans effet (T et U = 0)	BCS, 4	TEST1	
de même ici	LI, 0	X'1000'	
Dépouillement de l'instruction Branchement à (SSIØ) : Adresse E/S = CRA03 ⇒ Test sur T T = 0 ⇒ T := 15 (par exemple) UECØC:=(0) = 1000 CC:= 0 on passe à l'exécution de (GES) : T ≠ 0 ⇒ T := T -1 = 14	SIØ, 0	CRA03 →	Reconnaissance du périphérique sont état est "libre" ⟹ on le met à l'état occupé, on initialise le compteur de commandes avec (0) c.a.d. 1000 - Départ E/S
	positionnement du code de condition à 0		début de l'exécution de la lecture sur cartes
T ≠ 0 ⇒ T := T -1 = 13	BCS, 4	ERD	
T ≠ 0 ⇒ T := T -1 = 12	instruction n° 1		
⋮	⋮		
T ≠ 0 ⇒ T := T -1 = 3	instruction n° 10		
Dépouillement de l'instruction Branchement à (STIØ) Adresse E/S = CRA03 ⇒ Test sur T T ≠ 0 ⇒ CC := 4 Branchement à (GES) : T ≠ 0 ⇒ T := T -1 = 2	TIØ, 0	CRA03 →	envoi à l'U. C. de l'état du lecteur : occupé
	positionnement du CC : 0100		
T ≠ 0 ⇒ T := T -1 = 1	BCS, 4	TEST2	

Dépouillement de l'instruction Branchement à (STIØ) T ≠ 0 ⇒ CC := 4 Branchement à (GES) : T ≠ 0 ⇒ T = T -1 = 0 T = 0 ⇒ Test sur UECØC	TIØ, 0	CRA03 →	envoi à l'U. C. de l'état du lecteur : occupé
	positionnement du code de condition : 0100		
UECØC = 1000 ⇒ Exécution de la lecture de cartes définie par le jeu standard de commandes			fin de l'exécution de la lecture sur cartes
	BCS, 4	TEST2	
Dépouillement de l'instruction Branchement à (STIØ) : Adresse E/S = CRA03 ⇒ Test sur T T = 0 ⇒ CC := 0	TIØ, 0	CRA03 →	envoi à l'U. C. de l'état du lecteur : libre
	positionnement du code de condition à 0		
	BCS, 4	TEST2	
	instruction n° 13		

CHAPITRE VI

ILM70-4 UTILISATION D'UN DISQUE COMME MEMOIRE SECONDAIRE

ILM70-4 possède le même jeu d'instructions que ILM70-3. On a ajouté à ILM70-3 une mémoire secondaire, c'est-à-dire un disque, et la possiblité de faire exécuter par l'unité d'échange des commandes ou des suites de commandes situées à n'importe quelle adresse paire en mémoire.

Cette machine permet l'étude des problèmes posés par la gestion d'un espace disque et la mise au point en Travaux Pratiques d'un mini-système effectuant cette gestion.

VI. 1 Description d'ILM70-4

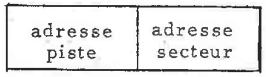
On trouve dans ILM70-4 tous les éléments d'ILM70-3. Un programme écrit pour ILM70-3, en particulier un assembleur, pourra être utilisé sur ILM70-4.

VI, 1. 1. Disque

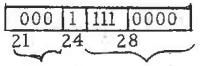
Le disque comprend 16 pistes. Chaque piste est composée de 16 secteurs (ou enregistrements) de 16 mots (64 octets). On a limité la taille de ces enregistrements pour éviter la réservation et l'initialisation de trop grands buffers lors de l'utilisation d'ILM70-4.

Les pistes et les secteurs sont numérotés de 0 à 15.

Une adresse disque est formée de 2 octets.



L'adresse d'entrée-sortie (cf. III. 4. 5) associée au disque est :



c'est-à-dire en héxadécimal '0F0'.

On pourra donc trouver dans les instructions SIØ et TIØ les 3 adresses d'entrées-sorties suivantes (exprimées en héxadécimal) :

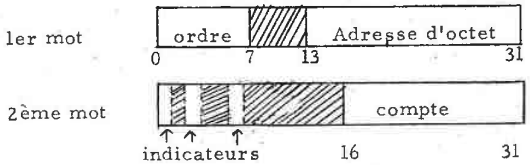
002	pour l'imprimante
003	pour le lecteur de cartes
0F0	pour le disque.

Une lecture ou une écriture sur disque relativement à un enregistrement donné est préparée à l'aide d'une commande sélection (voir VI. 1. 2) qui permet de préciser l'adresse disque de cet enregistrement. Après une opération d'entrée-sortie, on pointe vers le premier enregistrement qui n'a pas été concerné, même si on n'a utilisé qu'une zone partielle de l'enregistrement précédent. Si lors d'une écriture le nombre d'octets n'est pas un multiple de 64, le dernier enregistrement à écrire est complété par des 0.

VI. 1. 2 Programmation de l'unité d'échange

Elle se fait par des commandes ou des suites de commandes. Une commande est contenue dans 2 mots. L'adresse du 1er mot doit être pair.

a) Format d'une commande :



On trouve dans le 1er octet un ordre destiné au périphérique. Sur ILM70-4, un ordre peut être :

Ecriture	01
Lecture	02
Sélection	03.

L'adresse d'octet contenue dans les bits 13 à 31 du premier mot indique, dans le cas d'une lecture ou d'une écriture l'adresse du premier octet à transmettre.

Dans le cas d'une sélection, cette adresse indique le 1er octet d'une chaine de 2 octets. Le contenu de cette chaine est une adresse-disque (cf VI. 1. 1).

On trouve dans les bits 16 à 31 du 2è mot le compte des octets à transmettre, et, dans les bits 0, 2 et 6 de ce 2è mot 3 indicateurs.

Le bit 0 est l'indicateur d'enchainement de données (ED), le bit 2 est l'indicateur d'enchainement de commandes (EC), le bit 6 permet de demander l'inhibition du signal "longueur incorrecte".

b) Enchainement de données, enchainement de commandes.

L'indicateur d'enchainement de commandes permet de faire exécuter par l'unité d'échange des suites de commandes rangées séquentiellement en mémoire centrale sans que l'unité centrale ait à intervenir :

lorsqu'une commande où l'indicateur d'enchainement de commande est présent est terminée, l'unité d'échange va chercher la commande suivante en mémoire centrale et l'exécute.

L'indicateur d'enchainement de données permet, dans le cas d'une écriture, de regrouper dans un enregistrement le contenu de plusieurs zones de la mémoire ; dans le cas d'une lecture, de répartir le contenu d'un enregistrement dans diverses zones de la mémoire.

Lorsqu'une commande où l'indicateur d'enchainement de données est présent, est terminée, l'unité d'échange va lire en mémoire centrale la commande suivante. La zone ordre de cette commande est ignorée. L'opération précédente se poursuit sur la nouvelle chaine d'octets définie par la commande. Dans le cas où les 2 indicateurs d'enchainement de données et d'enchainement de commandes sont présents, un enchainement de données est exécuté .

c) Inhibition du signal "longueur incorrecte"

Dans le cas où le compte d'octets d'une commande ne correspond pas à la taille de l'enregistrement physique du périphérique, le programme est interrompu et le diagnostic d'erreur

'LONGUEUR INCORRECTE'

est imprimé.

Dans le cas du disque, on peut inhiber ce signal en positionnant le bit 6 du deuxième mot de la commande à 1. Ceci permet de ne lire un enregistrement que partiellement. Dans le cas d'une écriture partielle, le reste de l'enregistrement est complété par des 0. Cette inhibition est impossible dans les commandes relatives au lecteur de cartes où à l'imprimante. Dans ce cas, le bit 6 du 2ème mot est ignoré.

d) Exemple de programmation de l'unité d'échange

Soit ØCT1, ØCT2, ØCT3, ØCT4 4 adresses d'octets. On trouve à ces adresses :

ØCT1	X'0003'
ØCT2	'A B C ...'
ØCT3	'D E ...'
ØCT4	'F G H I ...'

Si on lance, par une SIØ, l'exécution des commandes suivantes :

Ordre	Adresse d'octet	Indicateurs	Compte
Sélection	ØCT1	EC	2
Ecriture	ØCT2	ED, LI	3
	ØCT3	EC, LI	2
Ecriture	ØCT4	LI	4

(EC, ED, LI indiquent respectivement l'indicateurs d'enchainement de commandes, d'enchainement de données et d'inhibition du signal de longueur incorrecte).

L'unité d'échange effectuera les opérations suivantes :

- sélection sur le disque du secteur n° 3 de la piste n° 0
- écriture dans ce secteur des 3 octets ABC puis des octets DE. Le secteur est complété par des 0.
- Ecriture dans le secteur n° 4 de la piste 0 des 4 octets FGHI suivis de 0.

VI. 2 Simulation d'ILM70-4

Le programme de simulation d'ILM70-4 s'appelle NIV4. C'est le programme NIV3 légèrement modifié.

VI. 1. 2 Représentation du disque

Le disque d'ILM70-est représenté par un fichier à clés sur disques du 10070 dont le nom est DISC. Chaque enregistrement comporte 64 octets. La clé associée à un enregistrement donné est une chaine de 3 octets Elle est précédée d'un octet où se trouve 3 qui est le nombre d'octets de la clé ; dans les 2 octets suivants se trouve l'adresse disque ILM70 associée à l'enregistrement ;

l'octet suivant contient zéro et est inutile (il s'agit d'une contrainte de programmation du 10070). La clé du prochain enregistrement à lire où écrire sur le disque simulé est repérée par le contenu du mot DKEY.

Le temps d'exécution d'une opération d'entrée-sortie sur le disque est représenté par l'"indicateur de temps d'attente" S (cf V. 2. 3. 2).

VI. 2. 2 Simulation de l'unité d'échange :

3 sous-programmes de simulation de l'unité d'échange ont été écrits. Ils sont associés à un périphérique. Ils sont mis en œuvre lorsqu'un indicateur de temps d'attente s'annule (cf. V. 2. 3) et lorsque l'adresse de commande contenue dans le registre zéro ne correspond pas à un des 2 jeux standard de commandes (cf. III. 4. 5).

Ces 3 sous-programmes sont indépendants. Ce sont des interprètes ; ils utilisent un compteur de commande qui contient l'adresse 10070 de la prochaine commande à exécuter et, un registre de commande qui contient la commande en cours. Chaque fois qu'une nouvelle commande est chargée dans le registre de commande, le compteur de commande est incrémenté de 2. Au moment du branchement à un de ces sous-programmes les registres d'ILM70 sont rangés dans les mots MEM à MEM+15. Ils sont restaurés au moment du retour dans NIV4.

VI. 2. 2. 1 Description du sous-programme SIMD : interprétation des commandes d'unité d'échange relatives au disque.

SIMD1 : Sauvegarde des registres ILM70 - Initialisation du compteur de commande.

SIMD2 : Chargement de la commande à exécuter - Progression du compteur de commande.

SIMD3 : Décodage de la commande et translation de l'adresse d'octet.

SIMD4 : Aiguillage, selon le contenu de la zone ordre vers un des 3 traitements suivants :

SIMDL : Simulation d'une opération de lecture.

SIMDE : Simulation d'une opération d'écriture.

SIMDS : Simulation d'une sélection.

On trouve également dans SIMD 2 sous-programmes internes :

SIMD. 5 : Traitement d'un enchainement de données

SIMD. 6 : Progression de DKEY.

Et le traitement suivant :

SIMD. 7 : Test et traitement d'un enchainement de commandes.

SIMD1 : Initialisation du compteur de commande :

à la rencontre d'une instruction SIØ, on a rangé le contenu du registre 0 dans ce compteur. On multiplie ce nombre par 2 et on le translate de MEM : on obtient ainsi l'adresse 10070 du premier mot de la première commande à exécuter.

SIMD3 : Décodage de la commande et translation de l'adresse d'octet :

le décodage de la commande donne comme résultat :

- un mot appelé INDIC qui contient la zone indicateur
- une adresse d'octet translatée de 4 x MEM qui sera notée ØCT
- un entier C qui indique le nombre d'octets concernés par la commande.

Description des 2 sous-programmes utilisés par SIMD :

SIMD5 : Traitement d'un enchainement de données.

Ce sous-programme décode la commande repérée par le compteur de commande

en ignorant sa zone ordre. Les résultats sont les mêmes que ceux d'un décodage normal (SIMD3) :

INDIC contient la zone indicateur

ØCT est l'adresse d'octet translatée de 4 x MEM

C est le compte d'octets.

Le compteur de commande est incrémenté de 2

SIMD6 : Progression de DKEY.

Ce sous-programme modifie la clé contenue dans le mot DKEY de la manière suivante : l'octet indiquant l'adresse du secteur est incrémenté de 1. S'il est égal à 16, on l'annule et on augmente de 1 l'octet indiquant la piste.

Description du traitement de l'enchainement de commande :

SIMD7 : Test et traitement d'un enchainement de commande.

Si l'indicateur d'enchainement de commande est présent dans INDIC, on recommence le traitement à partir de SIMD2. Sinon, on a exécuté toutes les commandes relatives à l'instruction SIØ et on retourne dans NIV4 après avoir restauré les registres ILM70.

Description de la simulation d'une opération de lecture : SIMDL

SIMDL1 : Lecture dans le fichier DISC de l'enregistrement (360 octets) repéré par la clé DKEY - I = 0 (I indique le nombre d'octets du tampon déjà rangés en mémoire).

SIMDL2 : Appel du sous-programme SIMD6 de progression de DKEY.

SIMDL3 : Si C > (360-I), on exécute SIMDL. 4, sinon SIMDL. 5.

SIMDL4 : Rangement de la chaine des (360-I) derniers octets du tampon à l'adresse ØCT. Mise à jour de la commande :

$$C = C - (360-I)$$

$$ØCT = ØCT + (360-I)$$

Recommencer SIMDL.

SIMDL. 5 : Fin de simulation de commande.

SIMDL5 est composé de :

SIMDL51 : Rangement de la chaine de C octets commençant au $I^{è}$ octet du tampon à l'adresse ØCT. I = I+C . La commande est alors terminée.

SIMDL52 : Test de l'indicateur d'enchainement de données. S'il est absent, on va exécuter SIMD7. Sinon, on appelle le sous-programme SIMD5 qui décode la nouvelle commande, et on recommence le traitement en SIMDL3.

SIMDE : Simulation d'une opération d'écriture.

SIMDE1 : Construction d'un tampon de 360 octets.

SIMDE2 : Ecriture de ce tampon dans l'enregistrement du fichier DISC repéré par DKEY.

SIMDE3 : Appel de SIMD6 : progression de DKEY.

SIMDE4 : La commande en cours est-elle terminée (C = 0 ?). Sinon retour en SIMDE1. Si oui SIMD7 :(Traitement de l'enchainement de commande).

Description de SIMDE1 :

SIMDE11 : I = 0. I représente le nombre d'octets déjà rangés dans le tampon.

SIMDE12 : Si C > (360-I) on exécute SIMDE13, sinon SIMDE14.

SIMDE13 : Rangement de la chaine de (360-I) octets commençant en ØCT à partir du I^{è} octet du tampon.
Mise à jour de la commande :

$$C = C - (360-I)$$
$$ØCT = ØCT + (360-I)$$

on passe ensuite à SIMDE2.

SIMDE14 : Rangement de la chaine de C octets commençant en ØCT à partir du I^{è} octet du tampon. I = I+C . C = 0.

SIMDE15 : Test et traitement de l'enchainement de données : si l'indicateur d'enchainement de données est présent, appel de SIMD5 et recommencer SIMDE12, sinon, mettre à 0 les (360-I) derniers octets du tampon et passer à SIMDE2.

SIMDS : Simulation d'une opération de sélection :

SIMDS1 : Rangement de la chaine de 2 octets commençant en ØCT à l'adresse DKEY+1.

SIMDS2 : Si l'indicateur d'enchainement de données est présent, on appelle SIMD5 et on recommence SIMDS, sinon, on passe à SIMD7.

VI. 2. 2. 2 Description du sous-programme SIMC : Interprétation des commandes d'unité d'échange relatives au lecteur de cartes.

SIMC1 : Sauvegarde des registres ILM70 - Initialisation du compteur de commande.

SIMC2 : Chargement dans le registre de commande de la commande à exécuter - Progression du compteur de commande.

SIMC3 : Décodage de la commande.

SIMC4 : Lecture d'une carte, et rangement du résultat de la lecture à l'adresse d'octet résultat du décodage.

SIMC5 : Test des indicateurs.

Dans la partie SIMC3 on trouve :

SIMC3.1 : Test de la zone ordre. Si elle est différente de la lecture il y a impression du diagnostic

'COMMANDE UE INCORRECTE'

et abandon du programme ILM70.

SIMC3.2 : Test de la zone compte : si elle est différente de 80, il y a impression du diagnostic

'LONGUEUR INCORRECTE'

et abandon du programme ILM70.

SIMC3.3 : Translation de l'adresse d'octet.

Dans la partie SIMC5, on effectue les traitements suivants :

Si l'indicateur d'enchainement de données est présent, on charge la nouvelle commande, on incrémente de 2 le compteur de commande et on recommence SIMC3.2 - Sinon, si l'indicateur d'enchainement de commande est présent, on recommence SIMC2. Si aucun indicateur n'est présent, on retourne dans NIV4 après restauration des registres d'ILM70.

Le sous-programme SIMI, qui interprète les commandes d'unité d'échanges relatives à l'imprimante est identique, à quelques détails près à SIMC :

Les tests effectués au décodage de la commande sont différents car la zone ordre doit indiquer une écriture et le compte doit être égal à 132.

L'opération d'entrée-sortie effectuée est une impression.

BIBLIOGRAPHIE

J. ANDRE

Introduction aux principaux concepts de Simula 67.

ESD Control Data - Puteaux 1970.

S. ARCANGELI, G. ARGILAS, G. BAZERQUE, J. BRUSSET

Simulation de l'unité centrale de la calculatrice CAE 10070 sur l'ordinateur IBM 7044.

A. F. I. R. O. 6ème Congrès - Paris 1967.

J. ARSAC

Les systèmes de conduite des ordinateurs.

DUNOD - Paris 1968.

J. P. CRESTIN

Enseignement de la programmation au moyen de 3 machines pédagogiques A, B, C réalisées sur le système CAE 510 multipupitre.

Manuel d'utilisation - Université de Toulouse - Faculté des Sciences.

P. DEBRAINE

Machines de traitement de l'information. Tome II : Programmation Principes et langages d'assemblage.

Masson - Paris 1969.

D. D. MAC CRACKEN

Programmation des calculatrices numériques.

Dunod - Paris 1960.

L. NOLIN

Formalisation des notions de machine et de programme.

Gauthier-Villars - PARIS 1969.

A. PROFIT

Structure et technologie des ordinateurs - Chapitre II : Structures algorithmiques et organisation des calculateurs.

Armand Colin - 1970.

READER

Applications of Decision Tables.

Hermann - Auerbach Publishers Inc. 1971.

E. D. REILLY, Jr., and F. D. FEDERIGHI

The elements of digital computer programming.

Holden-Day, Inc. San Francisco - California 1968.

Moniteur BPM 10070 - Manuel d'utilisation - C900954C,

CII - Louveciennes.

Ordinateur 10070 - Manuel d'exploitation - NT 2444,

CII - Louveciennes.