CEA-R-3588

PREMIER MINISTRE

COMMISSARIAT A L'ENERGIE ATOMIQUE

CEA-R-3588

9.0

ADRESSAGE SYMBOLIQUE ABSOLU STRUCTURE FACILITANT LE TRAVAIL EN PARTAGE DE TEMPS

par

Pierre DEBRAINE

Centre d'Etudes Nucléaires de Saclay

Rapport CEA-R-3588

1968

Fa

SERVICE CENTRAL DE DOCUMENTATION DU C.E.A

C.E.N-SACLAY B.P. n°2, 91-GIF-sur-YVETTE-France

CEA-R-3588 - DEBRAINE Pierre

ADRESSAGE SYMBOLIQUE ABSOLU,
STRUCTURE FACILITANT LE TRAVAIL EN PARTAGE
DE TEMPS

Sommaire. - Le partage de temps des ordinateurs demande la réalisation d'un certain nombre de conditions, en particulier, un chargement dynamique efficace des programmes et des données. Cet article indique une méthode de pagination réalisant les liaisons avec un minimum de consultations de tables. Le principe consiste à utiliser une mémoire associative composée de registres pour appeler les blocs de la mémoire physique, l'adresse de bloc étant donnée par concaténation d'un n° de fichier (logé dans un registre de base) et d'un n° de page (logé dans l'instruction proprement dite). La position à l'intérieur du bloc est donnée par un déplacement logé dans l'instruction. Un second registre de base, associé

CEA-R-3588 - DEBRAINE Pierre

ABSOLUTE SYMBOLIC ADDRESSING,
A STRUCTURE MAKING TIME-SHARING EASIER

Summary. - Time-sharing of computers asks for a certain number of conditions, particularly an efficient dynamic loading of programs and data. This paper indicates a paging method making linkages with a minimum of table-looking operations. The principe is to use associative memory registers for calling blocks of physical memory, the block address being given by the concatenation of a file number (located in a base register) and a page number (located in the instruction proper). The position within the block is given by a displacement located in the instruction. A second associated base register contains the local part (page number + displacement) of the base address. This extended

au premier, contient la partie locale (n° de page + déplacement) de l'adresse de base. Ce système d'extension de registre de base permet l'exécution de programmes appartenant à un très grand complexe sans introduire de pertes de temps. Les adresses sont fixées au moment de l'assemblage et les blocs peuvent être chargés n'importe où, pour exécution, sans modification.

Les différents problèmes liés à l'exécution de programmes complexes sont présentés dans ce cadre et l'on montre qu'ils sont facilement résolus par le système proposé dont la réalisation serait très facile à partir des structures d'ordinateurs existant actuellement.

1968 62 p.

Commissariat à l'Energie Atomique - France

---

base register system allows executing programs in a very large programming complex without loss of time. The addresses are fixed at assenbly time and the blocks can be loaded anywhere without modification for execution.

The various problems associated with the execution of complex programs are presented in this context and shown to be easily solved by the proposed system, the realization of which would be very easy starting from the computer structures existing now.

1968 62 p.

Commissariat à l'Energie Atomique - France

A partir de 1968, les rapports CEA sont classés selon les catégories qui figurent dans le plan de classification ci-dessous et peuvent être obtenus soit en collections complètes, soit en collections partielles d'après ces catégories.

Ceux de nos correspondants qui reçoivent systématiquement nos rapports à titre d'échange, et qui sont intéressés par cette diffusion sélective, sont priés de se reporter à la lettre circulaire CENS/DOC/67/4690 du 20 décembre 1967 que nous leur avons adressée, et qui précise les conditions de diffusion.

A cette occasion nous rappelons que les rapports CEA sont également vendus au numéro par la Direction de la Documentation Française, 31, quai Voltaire, Paris 7e.

PLAN DE CLASSIFICATION

1. APPLICATIONS INDUSTRIELLES DES ISOTOPES ET DES RAYONNEMENTS

2. BIOLOGIE ET MEDECINE

2. 1 Biologie générale
2. 2 Indicateurs nucléaires en biologie
2. 3 Médecine du travail
2. 4 Radiobiologie et Radioagronomie
2. 5 Utilisation des techniques nucléaires en médecine

3. CHIMIE

3. 1 Chimie générale
3. 2 Chimie analytique
3. 3 Procédés de séparation
3. 4 Radiochimie

4. ETUDES DU DOMAINE DE L'ESPACE

5. GEOPHYSIQUE, GEOLOGIE, MINERALOGIE ET METEOROLOGIE

6. METAUX, CERAMIQUES ET AUTRES MATERIAUX

6. 1 Fabrication, propriétés et structure des matériaux
6. 2 Effets des rayonnements sur les matériaux
6. 3 Corrosion

7. NEUTRONIQUE, PHYSIQUE ET TECHNOLOGIE DES REACTEURS

7. 1 Neutronique et physique des réacteurs
7. 2 Refroidissement, protection, contrôle et sécurité
7. 3 Matériaux de structure et éléments classiques des réacteurs

8. PHYSIQUE

8. 1 Accélérateurs
8. 2 Electricité, électronique, détection des rayonnements
8. 3 Physique des plasmas
8. 4 Physique des états condensés de la matière
8. 5 Physique corpusculaire à haute énergie
8. 6 Physique nucléoire
8. 7 Electronique quantique, lasers

9. PHYSIQUE THEORIQUE ET MATHEMATIQUES

10. PROTECTION ET CONTROLE DES RAYONNEMENTS. TRAITEMENT DES EFFLUENTS

10. 1 Protection sanitaire
10. 2 Contrôle des rayonnements
10. 3 Traitement des effluents

11. SEPARATION DES ISOTOPES

12. TECHNIQUES

12. 1 Méconique des fluides - Techniques du vide
12. 2 Techniques des températures extrêmes
12. 3 Méconique et outillage

13. UTILISATION ET DEVELOPPEMENT DE L'ENERGIE ATOMIQUE

13. 1 Centres d'études nucléaires, laboratoires et usines
13. 2 Etudes économiques, programmes
13. 3 Divers (documentation, administration, législation, etc...)

Les rapports du COMMISSARIAT A L'ENERGIE ATOMIQUE sont, à partir du n° 2 200, en vente à la Documentation Française, Secrétariat Général du Gouvernement, Direction de la Documentation, 31, quai Voltaire, PARIS VIIe.

The C.E.A. reports starting with n° 2 200 are available at the Documentation Française, Secrétariat Général du Gouvernement, Direction de la Documentation, 31, quai Voltaire, PARIS VIIe.

- Rapport CEA-R-3588 -

Centre d'Etudes Nucléaires de Saclay
Services Scientifiques

ADRESSAGE SYMBOLIQUE ABSOLU, STRUCTURE FACILITANT LE TRAVAIL EN PARTAGE DE TEMPS

par

Pierre DEBRAINE
Professeur à l'Institut National
des Sciences et Techniques Nucléaires

- Août 1968 -

ADRESSAGE SYMBOLIQUE ABSOLU
STRUCTURE FACILITANT LE TRAVAIL
EN PARTAGE DE TEMPS

1 - RAPPEL DES CONDITIONS DU TRAVAIL EN PARTAGE DE TEMPS

Un ordinateur travaille en partage de temps quand plusieurs utilisateurs peuvent effectuer des tâches, demandant éventuellement une communication avec l'ordinateur, dans des conditions de simultanéité apparente, par suite d'un multiplexage dans le temps de l'unité centrale.

Chaque tâche a la commande à tour de rôle ; les programmes relatifs à la tâche qui travaille à un instant donné sont dans la mémoire centrale (éventuellement avec d'autres, relatifs à d'autres tâches) et les moyens de traitement désirés sont affectés à ce travail. La simultanéité apparente vient de ce que la commande est donnée à chaque tâche, avec une fréquence suffisamment grande, pendant des temps très courts. Cela permet à chaque utilisateur d'avoir l'impression d'utiliser l'ordinateur à plein temps ; en fait, cela n'a d'intérêt que dans le cas où l'utilisateur dialogue avec l'ordinateur et, c'est la cadence des échanges qui fixe la fréquence désirable des retours : les conditions

correctes sont réalisées quand, compte tenu de la lenteur relative des périphériques, l'utilisateur n'attend pratiquement pas pour envoyer ses messages et obtenir les réponses.

Il y a lieu de remarquer que l'utilisateur peut être un équipement automatique dont la rapidité des échanges avec l'ordinateur peut être, ou doit être, plus grande. S'il y a plusieurs utilisateurs de ce dernier type, la solution indiquée plus haut peut ne pas convenir par suite d'une rotation trop rapide ; une solution, dans ce cas, est d'utiliser un ordinateur en partage de temps pour chaque tâche relative à un équipement prioritaire et, de former un réseau de façon à utiliser l'ensemble des ordinateurs dans un système de multitraitement pour les tâches courantes, sans mettre en cause le travail des tâches prioritaires. Ce système est plus complexe, mais exige que chacun des ordinateurs travaille en partage de temps à son niveau.

Par suite, les méthodes permettant de faciliter le travail en partage de temps présentent un grand intérêt pour les utilisations à venir.

Le travail en partage de temps présente un problème de gestion important de l'ordinateur. Un système de programmes de service appelé superviseur doit assurer la répartition du travail, affecter les équipements et veiller à ce que les conditions nécessaires à l'exécution des tâches soient remplies. Le travail du superviseur s'effectue avec l'aide de répertoires et tables ou listes enregistrant la situation à tout moment.

Le partage de temps introduit des contraintes qui n'existent pas dans les utilisations habituelles où l'ordinateur exécute complètement une tâche avant de passer à la suivante.

Tout d'abord, plusieurs tâches étant en cours au même moment, les programmes correspondants doivent exister simultanément en mémoire centrale ; ceci reste vrai, même si l'on en met en garde dans une mémoire auxiliaire, car les programmes de la tâche suivante doivent être en mémoire centrale si l'on veut une passation de commande efficace. Il y a donc nécessité d'avoir un mécanisme de protection mémoire empêchant une tâche de démolir les programmes d'une autre tâche ou ceux du superviseur.

Les tâches étant interrompues avant d'être achevées, il y a lieu de mettre en garde l'état de l'ordinateur au moment de l'interruption et de rétablir cet état lors de la reprise. Cette opération est à la charge du superviseur ; celui-ci est amené à mettre en garde des programmes et des blocs de données en mémoire auxiliaire, pour faire de la place en mémoire centrale aux programmes relatifs à la nouvelle tâche à exécuter. Il faut, bien sûr, qu'il prenne note des localisations de ces différents programmes pour la reprise mais, de plus, il faut procéder au chargement dynamique des programmes dans des positions dépendant de la situation actuelle. En particulier, un même programme peut être implanté à des endroits différents de la mémoire centrale au cours de ses différents stages dans celle-ci. Le chargement dynamique est également nécessaire pour permettre l'appel des programmes au fur et à mesure des besoins, de façon à ne pas surcharger inutilement la mémoire centrale (ceci est possible car, par suite du partage de temps, le temps d'approvisionnement n'est pas perdu par l'ordinateur). Ce problème de chargement dynamique dans différentes parties de la mémoire est très important et est mal résolu par la technique du chargement translatable, qui donne le maximum de souplesse dans les utilisations traditionnelles. Nous développerons plus largement cette question dans ce qui suit.

L'utilisation des dispositifs d'interruption de programmes dans les ordinateurs modernes est, par contre, une facilité pratiquement

indispensable pour la réalisation du partage de temps, par suite de la possibilité que cela donne à l'ordinateur de réagir à des signaux extérieurs (ou intérieurs). Nous supposons ce mécanisme bien connu du lecteur.

Nous allons, dans ce qui suit, étudier le problème de l'adressage de façon à voir comment on peut réaliser le chargement dans les conditions les plus satisfaisantes.

Ceci nous amènera à une structure d'ordinateur sur laquelle nous verrons que la protection mémoire est réalisée et que l'exécution peut se faire sur des programmes chargés sans modification, quelle que soit l'implantation choisie par le superviseur.

Nous analyserons, ensuite, les différentes opérations qui peuvent se rencontrer en cours d'exécution des tâches, en particulier, nous verrons le problème des liaisons entre programmes assemblés séparément, ainsi que celui de la transmission des arguments dans ce cas.

Nous envisagerons un certain nombre de possibilités augmentant la puissance de travail de l'ordinateur (en particulier, l'utilisation de programmes communs et de programmes récursifs, la création de tableaux et de listes enchaînées dynamiques).

Nous verrons, enfin, comment le superviseur doit être conçu pour répartir les travaux et quels sont les moyens d'information et de gestion qu'il doit posséder pour diriger un grand système d'exploitation.

2 - ADRESSAGE LOGIQUE

La plupart des instructions des ordinateurs font appel à des opérandes spécifiés par leur adresse en mémoire au moment de l'exécution. Plusieurs méthodes existent pour spécifier ces adresses :

1°) - Mettre dans les instructions les adresses réelles, dites absolues, des opérandes de façon à les utiliser lors de l'exécution. Cette méthode présente l'inconvénient d'obliger à un chargement rigide dans des positions décidées lors de l'écriture du programme.

2°) - Utiliser des adresses symboliques transformées par l'assembleur en adresses relatives à la position de tête du programme. Lors du chargement, sachant où va se loger la tête du programme, il est possible de modifier toutes les adresses qui le nécessitent en ajoutant l'adresse d'implantation d'une façon systématique. Cette addition peut se faire, au moment de l'exécution seulement, par l'emploi d'un registre de translation contenant l'adresse d'implantation, fixée au moment du chargement. Cette méthode d'adressage dit translatable est très employée et facilite beaucoup l'utilisation simultanée de programmes et de sous-programmes assemblés séparément.

L'utilisation de registres de base relève du même esprit, ces registres jouent le rôle de registres de translation. L'adresse dans ce cas s'obtient par addition automatique, au moment de l'exécution, d'un déplacement, contenu dans l'instruction, au contenu du registre de base dont le numéro est également logé dans l'instruction considérée.

Dans le cas du partage de temps, la méthode de l'adressage translatable manque de souplesse, compte tenu des multiples chargements

successifs qui peuvent être nécessaires au cours de l'exécution de chaque tâche individuelle. Les difficultés viennent de la manière dont l'application de l'ensemble des adresses logiques (les adresses symboliques) dans l'ensemble des adresses physiques (les positions de mémoire) est réalisée. La correspondance est établie de façon que l'adresse appelée soit celle de la cellule considérée, soit directement, par suite de modifications introduites lors du chargement, soit indirectement, par suite de l'entrée en jeu du registre de translation lors du fonctionnement du dispositif d'appel de la mémoire.

Dans l'assemblage translatable, on peut dire que chaque adresse se compose en fait du nom du programme et de l'adresse relative dans ce programme. Le nom du programme reste implicite et n'est connu que du chargeur qui modifie les adresses appelées en fonction de l'implantation qu'il choisit.

3°) - On peut envisager de réaliser l'assemblage en affectant à chaque symbole une adresse constituée par un numéro de programme ou numéro de fichier, un numéro de bloc et un numéro de ligne suivant la technique dite de la séparation en pages. Cette technique proposée par l'équipe ATLAS de l'Université de Manchester et par J.B. DENNIS du M.I.T. permet, grâce à l'utilisation de tables de liaison, certaines étant réalisées d'une façon câblée sous forme de mémoires associatives, d'établir une relation entre les adresses appelées et les adresses d'implantation réelles. Dans les techniques proposées jusqu'ici, le numéro du programme (ou son nom) restait relativement implicite et les liaisons étaient établies par édition de liaisons avant l'exécution de façon à former des segments destinés à travailler ensemble. De nombreux travaux ont été faits dans ce domaine, les principaux sont indiqués dans la bibliographie située en fin d'exposé.

Nous voudrions indiquer dans ce qui suit une solution qui nous paraît remplir les conditions désirables sans introduire de modifications très profondes dans la structure des ordinateurs. Chemin faisant nous verrons plus en détail certains des problèmes qui se posent dans ce genre d'exploitation.

Le but cherché est d'arriver à exécuter tous les programmes formant une bibliothèque en les chargeant sans modification, quelle que soit leur implantation dans la mémoire centrale. Le point important à se rappeler est que le numéro de programme fait partie de chaque adresse.

Le numéro de programme est une manière de désigner celui-ci et doit être en relation biunivoque avec le nom symbolique correspondant. Un dictionnaire des fichiers peut être constitué de façon à permettre, soit par appel du symbole, soit par appel du numéro, de trouver l'implantation actuelle de chaque fichier qui peut contenir, soit un programme, soit des données.

L'intérêt de cette technique est de garder un adressage purement logique avec, à première vue, l'inconvénient d'introduire un numéro de fichier explicite, numéro pouvant être assez grand si l'on veut pouvoir constituer un système important. Cette dernière objection pourra être atténuée considérablement ainsi que nous le verrons plus loin.

Le point important est de pouvoir utiliser un dispositif permettant de réaliser l'application de la partie de l'ensemble des adresses logiques utilisées dans l'ensemble des adresses physiques, d'une façon commode et systématique. Nous verrons plus loin comment on peut utiliser dans ce but une mémoire associative, adressable directement, chargée au moment du chargement des blocs.

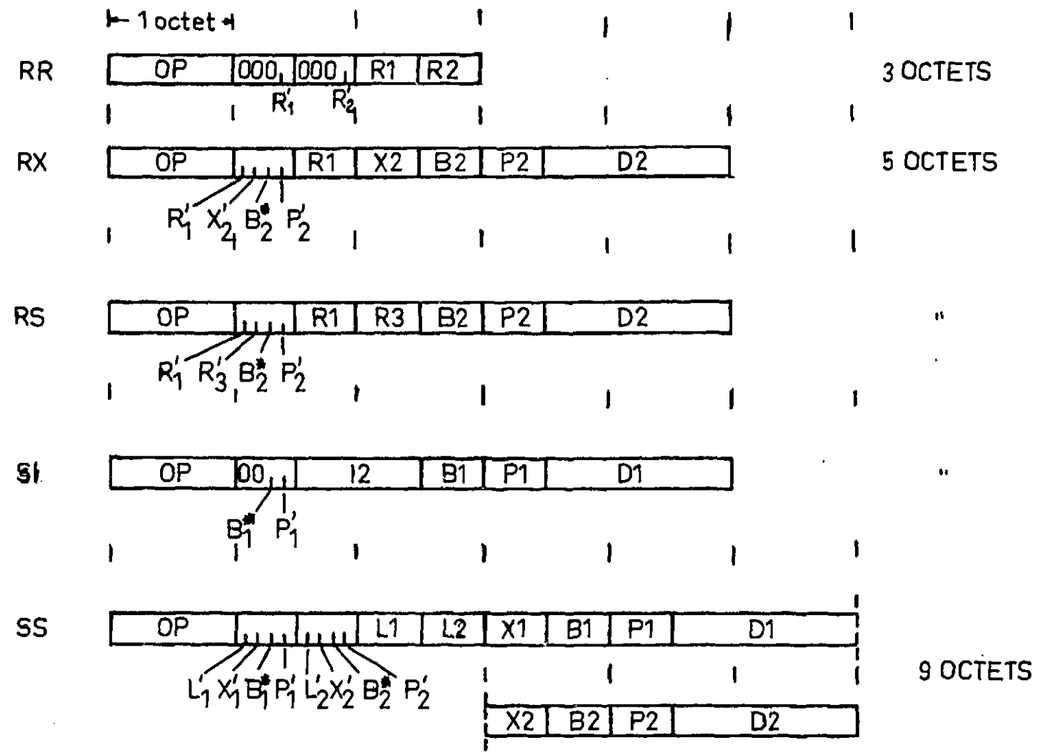
Le numéro de fichier (ou de programme) peut rester relativement implicite pendant l'exécution sous réserve qu'il reste logé dans un registre de base utilisé lors de l'appel de la mémoire centrale. L'un de ces registres de base peut jouer un rôle spécial analogue à celui du registre d'adresse d'instruction des ordinateurs traditionnels à simple adresse.

En réalité, nous allons envisager des paires de registres pouvant être utilisés groupés ou isolément, suivant les circonstances. Un numéro désignera la paire considérée ; dans les instructions qui ne font intervenir qu'un registre le n° seul indiquera le premier registre de la paire, le même n° avec apostrophe désignera le second ; dans les instructions qui font intervenir automatiquement les deux registres, le n° de la paire suffira ; dans les instructions qui peuvent faire intervenir le premier seul ou la paire, un astérisque suivant le n° indiquera cette deuxième utilisation.

Ce dernier cas s'applique aux registres de base qui fourniront, suivant les cas, le n° de fichier seul ou une adresse complète.

Afin de fixer les idées, nous utiliserons la structure d'instructions de l'I.B.M.360 (avec les mêmes codes opération pour désigner les instructions qui font pratiquement le même travail). Nous allons supposer qu'il y a I6 paires de registres généraux numérotées de 0 à 15_{10} (soit 0 à F_{16}). Les instructions vont être allongées (tout en restant dans le cadre de 3 longueurs différentes) de façon à donner la possibilité de désigner les combinaisons R' et R* , à côté de R.

Les différentes instructions peuvent avoir les modèles suivants :



Les zones P et D (4 digits et I2 digits) sont relatives à l'adresse locale : page et déplacement ; le digit P' permet d'amener le nombre de pages à 32. Les zones B,X et R désignent les registres généraux de 0 à 15 ; les digits B* , X' et R' indiquent l'utilisation des registres par paires ou isolément. Les digits L'_1 et L'_2 permettent d'augmenter les longueurs de zones, si nécessaire. La position de ces digits est choisie de façon à faciliter l'interprétation de l'instruction donnée sous forme hexadécimale par suite du découpage de 4 en 4 digits.

Avec ces conventions, les adresses locales P.D permettent de désigner 32 blocs de 4096 octets (1024 mots de 4 octets). Nous désignerons dans ce qui suit par P le groupe formé par le digit P' suivi des 4 digits P

de l'instruction. Le n° de fichier sera logé dans le premier registre d'une paire ou dans un mot, suivant les circonstances ; avec un mot de 32 digits, il est possible de désigner 2^{32} fichiers différents, soit plus de 4 milliards, ce qui est assez peu limitatif.

Une adresse symbolique absolue est composée de 3 éléments (3 configurations binaires de 32,5 et 12 digits, respectivement) :

n° de fichier, n° de page, déplacement

le groupement n° de page, déplacement sera appelé adresse locale tandis que celui du n° de fichier et du n° de page sera désigné comme adresse de bloc.

Normalement, l'adresse de bloc sera obtenue en accolant le contenu du premier registre de la paire de base avec la partie n° de page résultant de l'addition de l'adresse locale logée dans le registre de commande (contenant l'instruction), du contenu du registre d'index indiqué (s'il y en a un) et du contenu du deuxième registre de la paire de base (s'il y a un astérisque). Les éléments non indiqués n'interviennent pas et l'addition se fait modulo 2^{17} de façon que l'indexage permette le changement automatique de page dans les deux sens. Pour l'appel de l'instruction suivante, c'est le contenu des deux registres de la paire 0 qui est utilisé pour fournir l'adresse à employer.

L'adresse de bloc sera envoyée pour comparaison aux registres associatifs rapides RASR pour sélection du bloc physique à appeler ; le déplacement résultant des opérations d'indexage indiquées plus haut est transmis au bloc physique sensibilisé pour déclenchement de l'appel mémoire. Seule la cellule de RASR qui contient l'adresse de bloc cherchée fournit le signal de reconnaissance ; il suffit donc de charger cette cellule de l'adresse de bloc au moment du chargement de ce bloc logique

dans le bloc physique correspondant pour permettre cet appel.

Si aucun bloc physique n'est sensibilisé, un signal d'interruption redonne la commande au superviseur pour faire le nécessaire (lancement du chargement du bloc demandé dans un bloc physique si cela est autorisé et passage à un autre travail ou émission d'un message d'avertissement).

3 - MECANISMES GENERAUX DES INSTRUCTIONS

Dans ce qui suit, on va supposer que les blocs appelés se trouvent dans la mémoire centrale et peuvent être atteints par l'intermédiaire des registres associatifs rapides.

3.1 - Instructions de chargement

L'instruction de chargement d'un registre à partir de la mémoire centrale a la forme générale

L R1,P.D(X,B)

R1 est un registre unique (on peut avoir R1 ou R1')

((B),P.D+(X)+(B')) ⟶ (R1)

décrit le fonctionnement de cette instruction, les parenthèses désignant le contenu de l'élément indiqué à l'intérieur.

(B) est le contenu du registre de base primaire, c'est à dire le n° de fichier

P.D est l'adresse locale indiquée dans l'instruction,

(X) est le contenu du registre d'index éventuel (qui n'intervient pas si X = 0)

(B') est le contenu du registre de base secondaire (le deuxième de la paire) si l'on a B* dans l'instruction. S'il n'y a pas de registre de base indiqué (ou si c'est le registre 0) : (B)=(0).

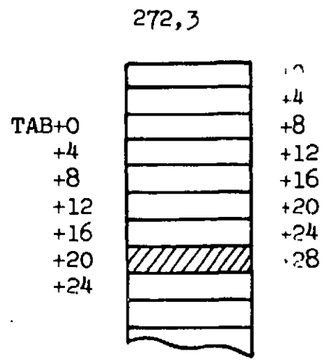
Supposons par exemple qu'un tableau du programme qui a actuellement la commande ait comme adresse de tête :

272,3.08

n° de fichier : 272
n° de page : 3
déplacement : 8

On veut charger le 6ième mot de ce tableau dans le registre n° 2
Ceci peut s'écrire :

L 2,3.08(7)



en supposant que l'on ait chargé préalablement 20 dans le registre n° 7 qui va servir d'index. Le registre 0 contient le n° de fichier : 272 et le registre secondaire 0' n'intervient pas (il contient l'adresse locale de l'instruction actuelle). On a :

((0),3.08+(7)) = (272,3.28) ⟶ (2)

(2) est le contenu du registre 2.
Ce qui charge bien le sixième mot de TAB dans le registre 2 (primaire).

L 2',3.08(7)

provoquerait ce chargement dans le registre secondaire 2. Tous les registres peuvent être utilisés de cette façon (il y a d'ailleurs intérêt à ce que le registre 0 soit chargé par une instruction privilégiée, c'est à dire utilisable uniquement par le superviseur, ce qui est facile à réaliser par reconnaissance du registre 0 comme premier opérande).

Comme deuxième exemple, prenons le même travail alors que le programme en cours d'exécution a comme n° de fichier : 2500.
((0)=2500). On suppose que l'adresse du tableau TAB = 272,3.08 a été logée dans la paire n° 5 de registres généraux. Les instructions de

chargement du 6ième mot de TAB dans le registre 2 peuvent s'écrire de différentes manières.

Si l'on utilise le registre de base comme registre pointeur ((5)=272,(5')=3.08) :

L 2,0.20(0,5*)

spécifie la position relative dans l'adresse locale (en fait, dans le déplacement, la partie page n'intervenant que pour des déplacements supérieurs à la taille d'une page) ;

L 2,0.00(7,5*)

permet un travail indexé, si le registre d'index 7 a été convenablement chargé.

L'adresse locale du tableau peut également être logée dans l'instruction, s'il y a indexage, en utilisant l'instruction.

L 2,3.08(7,5)

On peut également utiliser l'adresse locale de la position cherchée :

L 2,3.28(0,5)

Ces différentes instructions provoquent l'opération

(272,3.28) ⟶ (2)

Les instructions de chargement de registres ne posent pas de problème puisque tout se passe en interne et n'introduit pas de questions d'adressage dans la mémoire centrale.

LR 2,3'

provoque ainsi l'exécution (3') ⟶ (2).

Compte tenu de la nature des registres, il est commode d'introduire une instruction de chargement d'une paire de registres à partir

d'un double mot de la mémoire centrale :

LP 2,P.D.(X,B)

((B),P.D+(X)) ⟶ (2)

((B),P.D+(X)+4) ⟶ (2')

Le chargement multiple se fait de même en prenant les paires successives (éventuellement de façon incomplète). Ainsi :

LM 2,4,P.D(X,B*)

se traduirait par l'exécution des chargements suivants :

((B),P.D+(X)+(B') ⟶ (2)
((B),P.D+(X)+(B')+4) ⟶ (2')
((B),P.D+(X)+(B')+8) ⟶ (3)
((B),P.D+(X)+(B')+12) ⟶ (3')
((B),P.D+(X)+(B')+16) ⟶ (4)

Les limites peuvent être des registres primaires ou secondaires.

3.2 - Instructions de rangement

Elles sont strictement équivalentes aux précédentes mais fonctionnent en sens inverse. On définira, en particulier, une instruction STP R1,P.D(X,B) qui permet de mettre en garde une paire de registres de base.

3.3 - Instructions arithmétiques en virgule fixée

L'adressage s'effectue comme précédemment mais les paires de registres peuvent être utilisées pour les instructions qui font intervenir deux registres adjacents. Le registre primaire et le registre secondaire jouant le rôle des registres pair et impair successifs. La désignation est alors unique. Par exemple, dans une multiplication, après avoir chargé le multiplicande dans le registre 3', il suffit de spécifier la multiplication en utilisant le registre 3 pour que le produit se retrouve dans la paire (3),(3').

Ces instructions, ainsi qu'un grand nombre d'autres ne présentent pas de difficulté avec ce mode d'adressage.

3.4 - Instructions de branchement

Les instructions de branchement à une adresse explicite ne présentent pas de problème.

BC M,P.D(X,B*)

où M est le masque qui spécifie la condition, provoque, lorsque celle-ci est satisfaite, les opérations suivantes :

(B) ⟶ (0)
P.D+(X)+(B') ⟶ (0')

de même :

BC M,P.D(X,B)

donne :

(B) ⟶ (0)
P.D+(X) ⟶ (0')

Alors que

BC M,P.D(X)

donne

(0) ⟶ (0)
P.D+(X) ⟶ (0')

Ce qui effectue bien ce qui est demandé, d'une manière cohérente, dans les différents cas.

Lorsque l'adresse de branchement est spécifiée par un registre différent de 0, c'est toujours la paire qui est spécifiée.

L'instruction

BCR M,R (R≠0)

doit être interprêtée comme suit, si la condition est satisfaite :

(R) ⟶ (0)
(R') ⟶ (0')

tant que R est différent de 0.

Si l'adresse de branchement est interne, cette adresse doit être placée dans un registre secondaire avant d'exécuter l'instruction.

BC M,R'

qui se traduit par :

(R') ⟶ (0')
(0) restant inchangé

3.5 - Instructions de branchement avec liaison

Ces instructions qui permettent d'appeler les sous-programmes présentent deux formes.

Lorsque l'adresse est explicite, l'interprétation est toujours la même :

BAL R1,P.D(X,B*)

provoque la mise en garde de (B) et de P.D+(X)+(B'), l'envoi de (0) et (0') dans (R1) et (R1'), puis le chargement des quantités, mises en garde au début, dans (0) et (0'). Il y a donc mise en garde de la prochaine instruction actuelle dans la paire R1 et branchement à l'adresse indiquée. Si l'adresse est interne (B n'intervient pas) , le registre 0 jouant le rôle de B n'est pas modifié :

Avec l'instruction

BALR R1,R2

il y a deux formes possibles suivant que l'adresse de branchement est externe ou interne. Dans le cas de l'adresse interne, il faut

BALR R1,R2'

qui se traduit par :

(R2') ⟶ (AUX2)
(0) ⟶ (R1)
(0') ⟶ (R1')
(AUX2) ⟶ (0')
(0) reste inchangé

tandis que

BALR R1,R2

donne

(R2) ⟶ (AUX1)
(R2') ⟶ (AUX2)
(0) ⟶ (R1)
(0') ⟶ (R1')
(AUX1) ⟶ (0)
(AUX2) ⟶ (0')

AUX1 et AUX2 étant des zones de stockage intermédiaires existant dans la circuiterie. Par exemple, si l'adresse d'un sous-programme se trouve dans le double mot ADSP, l'appel peut se faire par

```
LP    15,ADSP
BALR  14,15
```

LP charge le n° de fichier dans 15 et l'adresse locale du point d'entrée dans 15' ; BALR met le n° de fichier de l'instruction dans 14 et son adresse locale dans 14'.

Le retour au programme appelant peut se faire par :

```
BCR   15,14
```

qui recharge les registres 0 et 0' à partir de 14 et 14'.

Dans le cas d'un sous-programme interne, on pourrait écrire :

```
LP    15,ADSP
BALR  14,15'
```

et revenir par

```
BCR   15,14'
```

mais il n'y aurait pas d'inconvénient à utiliser la même forme que dans le cas interne. On pourrait également utiliser L 15',ADSP+4 pour charger seulement 15' (à condition d'utiliser BALR 14,15').

3.6 - Instruction de chargement d'adresse

L'instruction de chargement d'adresse s'écrit

```
LA    R1,P.D(X,B*)
```

L'adresse à charger se compose de deux éléments (B) et P.D+(X)+(B') qui doivent aller dans R1 et R1' . On a donc :

(B) ⟶ (R1)
P.D+(X)+(B') ⟶ (R1')

Avec un certain nombre de cas particulier :

```
LA    R1,P.D(X,B)
```

donnant

(B) ⟶ (R1)
P.D+(X) ⟶ (R1')

```
LA    R1,P.D(X)
```

donnant

P.D+(X) ⟶ (R1')
(0) ⟶ (R1)

On peut également écrire

```
LA    R1',P.D(X)
```

qui donne

P.D+(X) ⟶ (R1')
(R1) sans changement

Cette dernière forme permet un chargement immédiat de registres secondaires, par

```
LA    R1',P.D
```

qui effectue P.D ⟶ (R1')

ou une évolution additive de registres secondaires par

```
LA    R1',P.D(R1')
```

qui effectue (R1')+P.D ⟶ (R1')

4 - ASSEMBLAGE

Nous avons vu que les adresses se notaient

P.D(X,B*)

dans les instructions quand on avait

(B) = n° fichier

P.D+(X)+(B') = adresse locale

(X) étant le contenu du registre d'index, P.D l'adresse locale intervenant dans l'instruction et (B') l'adresse locale de base (qui n'intervient pas s'il n'y a pas d'astérisque).

En réalité, lors de l'écriture du programme, on utilise des adresses symboliques qui peuvent être de trois types :

adresses internes
adresses extérieures
adresses externes

Les adresses internes sont celles qui sont définies dans le programme lui-même, le n° de fichier est celui du programme, l'adresse locale dépend de la localisation et peut être déterminée par l'assembleur utilisant son compteur d'adresses, comme dans l'assemblage traditionnel. Ces adresses utilisent le registre 0 comme registre de base. On peut remarquer que l'utilisation de 0* correspond à la désignation de l'adresse actuelle puisque 0' contient l'adresse locale de l'instruction actuelle.

Les adresses extérieures sont celles qui sont définies dans le programme actuel mais qui sont utilisées également par d'autres. Le nom du programme est une telle adresse définie par une pseudo-instruction START :

```
PROG    START
```

Les autres points d'entrée ou les variables extérieures proprement dites peuvent être définis par une pseudo-instruction ENTRY :

```
ENTRY   IN2, IN3, ALPHA, BETA
```

les différents symboles situés dans la partie variable étant des étiquettes du programme actuel.

Les pseudo-instructions START et ENTRY permettent à l'assembleur d'entrer les informations relatives à ces adresses dans le répertoire général. Celui-ci, organisé en listes enchainées, permet en entrant par le symbole PROG d'obtenir le n° de fichier affecté à ce programme, puis par le symbole ALPHA d'obtenir l'adresse locale de ce symbole. Le n° de fichier est affecté par l'assembleur par consultation d'une liste de numéros disponibles ; l'adresse locale est déterminée par l'assembleur au cours de son travail.

Les adresses externes sont celles qui sont définies dans un autre fichier mais utilisées dans le programme actuel (qui est également un fichier) ; l'autre fichier peut être un programme ou un fichier de données.

Ces adresses externes sont désignées par un symbole local, par exemple SYMB, définissant un double mot caractérisé par la pseudo instruction EXTRN.

```
SYMB    EXTRN   FCH,ALPHA
```

L'assembleur réserve un double mot à l'adresse SYMB et place dans le premier mot le n° de fichier FCH donné par consultation du répertoire général, dans le second l'adresse locale ALPHA donnée de même. En fait, ces renseignements peuvent ne pas exister déjà dans le répertoire quand on fait l'assemblage et ne pourront être introduits que lors d'un traitement ultérieur d'édition. Pour cela, l'assembleur construit une table de symboles externes, rappelant le symbole local SYMB les symboles FCH et ALPHA et leurs valeurs et associant un pointeur menant au double-mot logé à l'adresse SYMB (c'est à dire l'adresse locale de SYMB). Une marque mise en tête de l'adresse locale de SYMB indique que la mise en place a été faite dans le programme. Dans le répertoire général, on établit pour chaque fichier une liste des programmes qui l'utilisent ; ceci permettra une mise à jour en cas de modification du n° de ce fichier.

Chaque programme assemblé, muni de sa table de symboles externes est gardé sur un support auxiliaire (qui n'a pas besoin d'être accessible en permanence). Si tous les symboles externes sont marqués, le programme est édité et sa copie peut être placée en mémoire de stockage (qui doit être accessible en permanence si ce programme peut être appelé à tout moment). Les adresses périphériques (celles permettant de les appeler dans le stockage) sont mises en place dans le répertoire général associées au nom du fichier. Cette position est accessible par le n° de fichier pour les besoins du superviseur. Si les différentes pages d'un même fichier sont réparties dans le stockage, il faut introduire les adresses périphériques correspondantes dans le répertoire général avec appel par l'adresse de bloc. Si tous les symboles externes ne sont pas marqués, il faut faire imprimer la table des symboles externes pour savoir quels

sont les fichiers qui ne sont pas dans le répertoire ; après assemblage de ces fichiers manquants, une passe d'édition permet la mise en place des éléments désirés dans le programme traité, de façon à arriver à la version éditée indiquée plus haut. Le contrôle des marques permet de savoir si l'édition est complète et le programme utilisable.

A ce stade, les pages du programme sont chargeables dans des blocs physiques quelconques de la mémoire centrale, sans modification, pour exécution.

Il suffit que le superviseur charge dans la cellule correspondante des registres associatifs rapides l'adresse de bloc correspondante (qu'il connaît puisqu'il a établi l'ordre de chargement à partir de là).

Au cours du programme les appels à des symboles externes apparaissent sous deux formes :

1°) -chargement de registres de base des adresses complètes à partir des positions définies par EXTRN , chargées des configurations binaires voulues lors de l'édition.

2°)-instructions utilisant les registres de base précédemment chargés.

On peut envisager des instructions utilisant les symboles locaux de ces symboles externes, l'assembleur construisant l'instruction faisant intervenir le registre de base correspondant. Cette dernière possibilité serait particulièrement intéressante si l'assembleur tenait un compte de ce qui est chargé dans les registres de base de façon à mettre en place les instructions de chargement avant de composer une instruction utilisant un registre de base donné ; ce chargement ne se mettant pas en place si le registre de base contient déjà ce qui est nécessaire. Cette possibilité correspond au cas où le symbole utilisé dans l'instruction est

externe ce qui est indiqué dans la table des symboles lorsque l'on rencontre une pseudo-instruction EXTRN. Une solution mixte peut être envisagée, le programmeur écrivant les instructions de chargement des registres de base et l'assembleur tenant une table pour savoir quel est le registre de base utilisé avec chaque variable externe à l'instant considéré ; dans ce dernier cas, les instructions construites à partir du symbole externe feraient intervenir B* et non B seul. Cette solution mixte permet de garder le maximum de liberté pour la programmation.

5 - ORGANISATION D'ENSEMBLE (voir Fig. 1)

L'ordinateur est supposé constitué de la façon suivante :

1°) - Une unité centrale contenant 16 paires de registres généraux et les circuits permettant d'exécuter les instructions demandées suivant les indications du décodeur d'instructions logé sur le registre de commande RC.

2°) - Une mémoire centrale, constituée de blocs indépendants de 4096 octets (soit 1024 mots). Les différents blocs physiques ont leurs dispositifs d'accès propres et peuvent travailler en parallèle s'ils sont connectés temporairement à des dispositifs différents (unité centrale, canaux d'entrée-sortie).

3°) - Une unité de commande comprenant un registre de commande RC et un registre d'adresse d'instruction composé de la paire de base portant le numéro0 (le registre primaire 0 contient le n° de fichier ayant la commande, le registre secondaire 0' l'adresse locale de la prochaine instruction à exécuter). Un registre d'adresse mémoire général séparé en deux parties RAMT et RAMQ permet d'enregistrer l'adresse de bloc et le déplacement, respectivement. Le décodage de l'instruction permet de sensibiliser les registres de base et d'index et de faire passer leur contenu ainsi que celui de RC

contenant l'adresse locale dans l'additionneur d'indexage ADD de façon à délivrer les parties P et D à RAMT et RAMQ, respectivement. Le décodage de la marque * sur B met en service le registre B', l'absence de cette marque le laisse hors-service.

Les blocs mémoire, connectés à l'unité centrale, sont liés à un registre de données mémoire RDM, tandis que ceux liés à un canal d'entrée-sortie sont desservis par le registre tampon de celui-ci. Les registres d'adresses-mémoire locaux sont sensibilisés par les déplacements et par les lignes d'appel des blocs physiques venant du registre associatif rapide RASR que nous allons décrire maintenant.

Le registre associatif rapide est constitué d'un nombre de cellules égal au nombre de blocs physiques de la mémoire centrale qu'il dessert. Chacune de ces cellules individuelles peut être chargée, par le superviseur, de deux éléments différents : numéro de fichier (ou de programme) et numéro de page. Quand un appel mémoire est lancé, suivant une procédure sur laquelle nous reviendrons, le registre individuel contenant le numéro de fichier et le n° de page demandés (c'est à dire l'adresse de bloc demandée) logés dans RAMT, envoie un signal sur la ligne de sensibilisation correspondante de façon à sélectionner le bloc physique associé au bloc logique appelé.

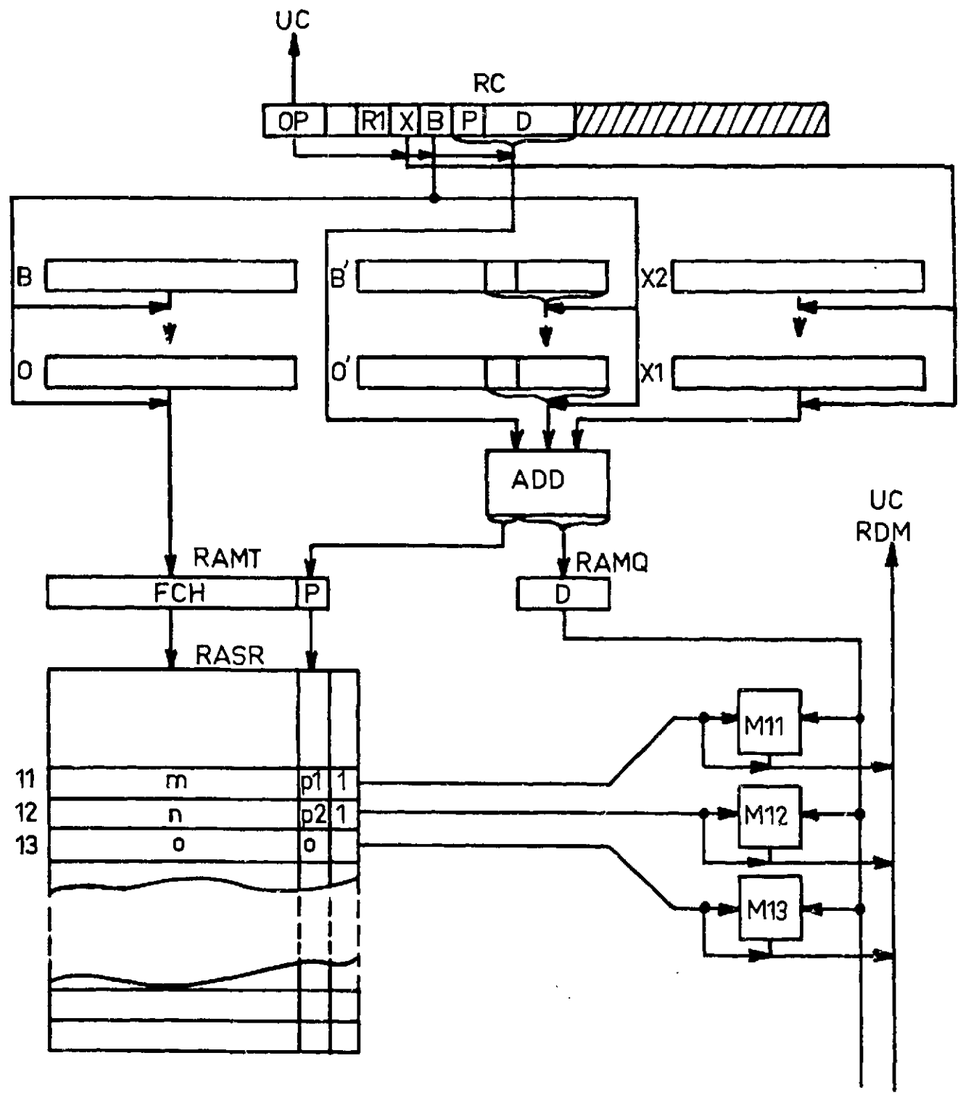


Fig. 1 - Organisation du bloc de commande

Le registre associatif est utilisé pour activer la reconnaissance du bloc physique intéressé. On peut l'envisager avec un ensemble de comparateurs jouant entre le registre d'entrée RAMT, contenant la demande, et chacune des cellules individuelles, permettant d'indiquer la cellule individuelle cherchée ; toutes les comparaisons s'effectuant en parallèle. On pourrait aussi avoir une comparaison séquentielle rapide, avec arrêt lorsque le résultat cherché est obtenu, ou une méthode mixte effectuant séquentiellement des comparaisons parallèles portant sur des cellules différentes du registre associatif. Le principe est toujours le même, mais le temps nécessaire et le volume de matériel à employer sont assez différents.

Il y a lieu de remarquer que les possibilités d'adressage de la mémoire centrale ne dépendent pas de la place occupée par l'adresse dans le mot, mais du nombre de cellules du registre associatif, chaque cellule correspondant à un bloc de 1024 mots, dans le cadre de l'exemple considéré.

- Des périphériques variés reliés à l'ordinateur par l'intermédiaire de canaux, comme dans les ordinateurs classiques, permettant le travail simultané des canaux et de l'unité centrale.

- Des mémoires de masse (disques ou autres) à accès relativement rapide sont importantes pour le stockage des fichiers de programmes et de données constituant le système.

La Fig. 2 montre un peu plus en détail comment se fait l'appel des blocs-mémoire par l'intermédiaire du registre associatif. Les cellules du registre associatif comportent des digits supplémentaires qui permettent d'enregistrer des marques facilitant le travail, nous en reparlerons plus loin.

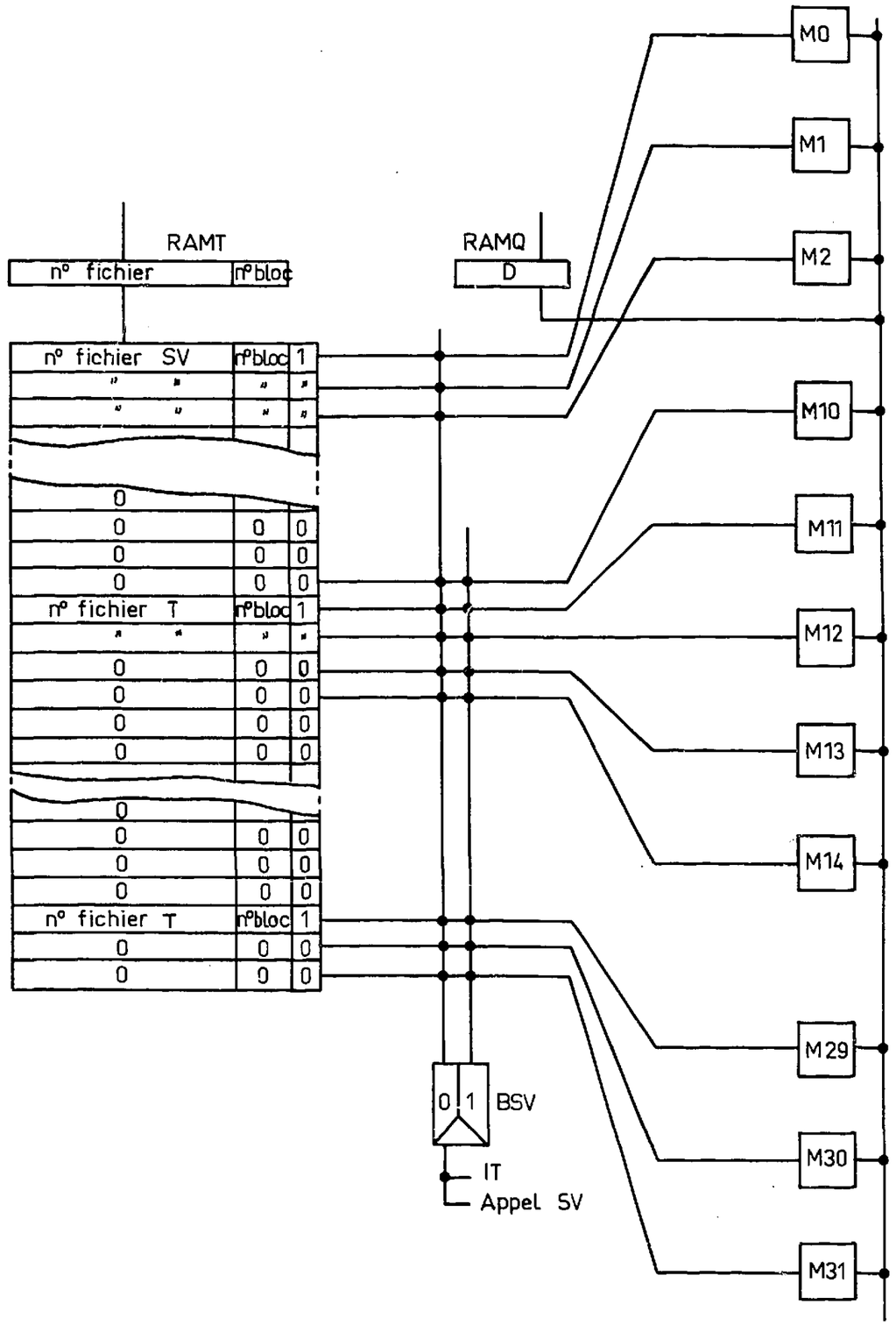


Fig. 2 - Appel individuel des registres mémoire par registres associatifs.

Un bistable BSV permet de changer de mode de fonctionnement (utilisateur ou superviseur) sur réception de signaux convenables. Ce bistable permet de bloquer l'accès aux blocs mémoire réservés au superviseur quand on est en mode utilisateur. En mode superviseur toutes les positions sont accessibles et les instructions privilégiées exécutables. La protection mémoire est assurée par le fait que seules les adresses de blocs autorisées sont dans RASR à l'exception de celles des blocs du superviseur ; la protection de ces dernières est assurée par le bistable BSV indiqué ci-dessus.

Bien évidemment l'ordinateur doit être muni de dispositifs d'interruption de programmes et de liaisons directes avec le monde extérieur. La mise en garde de l'état de la machine peut être faite au moyen d'un mot d'état programme comme dans le 360, augmenté pour pouvoir loger les parties utiles des registres 0 et 0' (il faudrait un triple mot pour loger le PSW, au lieu d'un double-mot) ; par contre, il serait intéressant d'avoir un plus grand nombre de lignes d'interruption hiérarchisées pour certaines applications.

6 - UTILISATION

6.1 - Appel d'une instruction

Supposons, pour commencer, qu'un programme ait la commande (le numéro de ce programme est dans le registre 0) et que les blocs dans lesquels il a à travailler soient chargés dans la mémoire centrale. Le registre 0' contient l'adresse locale de la prochaine instruction à exécuter. Dans la phase d'instruction, le n° de page de cette instruction est transmis à RAMT en même temps que le n° de fichier (adresse de bloc). Le bloc physique contenant cette instruction est sélectionné et le déplacement permet de lire l'instruction cherchée et de la transmettre

de RDM dans le registre de commande RC. Cette lecture peut se faire en plusieurs étapes et être pilotée par les digits de tête de l'instruction qui indiquent sa longueur.

A ce moment le décodeur de code opération sensibilise les circuits d'exécution de l'instruction appelée ; si c'est une instruction demandant un appel mémoire l'adresse est envoyée dans RAMT et RAMQ en tenant compte des registres de base mis en jeu et de l'indexage éventuel. Si l'adresse cherchée se trouve dans l'un des blocs physiques de la mémoire centrale, le déplacement sert à sélectionner l'adresse et le contenu se retrouve dans RDM pour utilisation (en une ou plusieurs étapes suivant les cas).

Lors de l'appel d'un bloc non présent en mémoire physique, ce qui est détecté par l'absence de sensibilisation d'un bloc par RASR, une interruption permet au superviseur d'étudier la situation. L'examen de RAMT lui donne l'adresse de bloc demandée ; la consultation de la table d'autorisation pour la tâche en cours lui permet de décider de l'appel du bloc demandé. Ce système réalise automatiquement la protection mémoire, par une opération qui n'est à effectuer qu'une fois par bloc. En effet, on ne peut avoir accès qu'aux blocs qui sont indiqués dans RASR, pendant l'exécution d'une tâche donnée. Si l'autorisation n'est pas donnée un message d'erreur, accompagné d'éléments d'information sur l'état actuel de la machine, est envoyé à l'utilisateur. Si le bloc demandé est présent dans la table d'autorisation, le superviseur lance un ordre d'entrée, après avoir affecté un bloc physique à ce bloc logique. L'appel est fait par le superviseur en consultant une fiche donnant l'implantation actuelle des blocs de cette tâche (en mémoire de stockage ou en mémoire auxiliaire) et en observant la table d'affectation de la mémoire. Voir plus loin l'établissement de la fiche de tâche.

Le superviseur passe alors la commande à la tâche suivante indiquée dans la file d'attente. Cette passation de commande est précédée d'une mise en garde de l'état machine dans le bloc de service réservé à la tâche interrompue et du chargement des registres à partir du bloc de service réservé à la nouvelle tâche. Nous reviendrons plus loin sur les différentes tables et listes que le superviseur utilisée et sur les blocs affectés à chaque tâche.

S'il n'y a pas de blocage, le registre 0' est majoré convenablement pour donner l'adresse locale de l'instruction suivante ; l'instruction située dans RC est exécutée et son achèvement déclenche la phase d'instruction suivante.

Si l'instruction demandée ne peut pas être exécutée par absence de l'élément voulu dans la mémoire physique, l'adresse de l'instruction est rétablie dans le registre 0' avant la mise en garde de façon que cette instruction soit exécutée de nouveau (c'est à dire soit effectivement exécutée) lors de la reprise. Dans le cas d'un branchement, c'est l'adresse de branchement qui reste dans la paire 0-0', pour la reprise après chargement du bloc correspondant. Dans la table d'affectation des blocs de mémoire, des digits indicateurs permettent de savoir si le bloc est affecté ou affecté et chargé. Normalement, la fin du chargement d'un bloc se traduit par une interruption permettant au superviseur de mettre à jour sa table d'affectation des blocs de mémoire, avant de redonner la commande à la tâche interrompue temporairement. Le superviseur peut s'inquiéter d'une demande répétée à un bloc qui a été demandé et qui n'est toujours pas chargé, en utilisant un compteur d'appels infructueux, dans la table d'affectation.

6.2 - Appel d'une variable interne

L'instruction symbolique

```
L       2,ALPHA
```


se trouve traduite par l'instruction

```
L   2,P.D.(0,0)
```

où P.D est l'adresse locale ALPHA. La variable étant interne le registre 0 contient le n° de fichier du programme en cours d'exécution. On a donc :

$$((0),P.D) \longrightarrow (2)$$

soit

$$(n°prog,P.D) \longrightarrow (2)$$

l'adresse de bloc est $[n°prog,P]$, le déplacement : D.

6.3 - Appel d'un tableau interne

On peut appeler un élément d'un tableau en utilisant le nom du tableau et le contenu d'un registre d'index. L'instruction symbolique

```
ST   2,TAB (3)
```

se trouve traduite par l'instruction

```
ST   2,P.D(3,0)
```

où P.D est l'adresse locale TAB. Le registre 0 contient le n° de fichier du programme en cours d'exécution. On a donc :

$$(2) \longrightarrow ((0),P.D+(3))$$

si $P.D+(3)=P'.D'$, on a :

$$(2) \longrightarrow (n°prog,P'.D')$$

l'adresse de bloc est $[n°prog,P']$, le déplacement: D'.

Le travail dans un tableau peut également se faire en logeant l'adresse du tableau dans un registre d'index utilisé comme registre pointeur et en travaillant en valeur relative.

LA 3',TAB

où TAB est une variable interne d'adresse [n°prog,P.D] provoque la construction de l'instruction.

LA 3',P.D

Ensuite l'instruction

ST 2,0.24(3')

provoquera l'exécution de

(2) ⟶ ((0),0.24+(3'))=(n°prog,0.24+P.D) soit

(2) ⟶ (n°prog,P.[D+24])

donc le rangement du registre 2 dans le 7ième mot du tableau TAB.

Si l'on veut pouvoir travailler en valeur relative mais avec un indexage possible, il faut utiliser un registre de base et un registre d'index.

Supposons par exemple que l'on veuille travailler en indexage par rapport à la valeur relative précédente. On peut utiliser la paire 3-3' comme registre de base.

LR 3,0 (0) = (n°prog) ⟶ (3)

LA 3',TAB P.D ⟶ (3')

en chargeant le registre d'index 4 et en utilisant l'instruction

ST 2,0.24(4,3 *)

on réalise

(2) ⟶ ((3),0.24+(4)+(3'))=((3),0.24+P.D+(4))

(2) ⟶ (n°prog,P.[D+24]+(4))

Les deux instructions de chargement peuvent se réaliser d'un seul coup en utilisant :

LA 3,TAB

qui devient

LA 3,P.D

6.4 - Appel d'un sous-programme interne

On suppose que les arguments sont des variables internes du programme appelant (donc également du programme appelé) dont les adresses locales sont logées dans un tableau interne LST d'adresse [n°prog,$P_1.D_1$] . Le sous-programme appelé est à l'adresse SP=[n°prog,$P_2.D_2$]. Les adresses locales des arguments, logées en LST, LST+4, LST+8, etc sont $P_3.D_3$, $P_4.D_4$, $P_5.D_5$, etc.

La séquence d'appel du sous-programme SP est :

LA 1,LST
LA 15,SP
BALR 14,15

Les quantités LST et SP sont reconnues comme symboles internes par l'assembleur qui construit les instructions :

LA 1,$P_1.D_1$ (0) ⟶(1),$P_1.D_1$ ⟶ (1')

LA 15,$P_2.D_2$ (0) ⟶(15),$P_2.D_2$⟶ (15')

BALR 14,15

Dans le sous-programme, pour appeler l'argument 1 dans le registre 7, on peut effectuer :

L 5,0.0(1') (n°prog,$P_1.D_1$)=$P_3.D_3$ ⟶(5)

puis L 7,0.0(5) (n°prog,$P_3.D_3$)=arg1 ⟶(7)

pour envoyer le contenu du registre 7' dans la position contenant l'argument 3, on peut utiliser :

```
    L      5,0.8(1')          (n°prog,P1.[D1+8])=P5.D5 ⟶ (5)
puis ST    7',0.0(5)          (7') ⟶ (n°prog,P5.D5)
```

Le retour au programme appelant peut se faire par

```
    BCR    15,14
```

qui rétablit l'adresse de l'instruction suivante à exécuter dans le registre 0'.

La liste d'adresses d'arguments peut être construite au moyen de constantes d'adresses (qui indiquent que les quantités indiquées sont des adresses locales).

On aura ainsi

```
LST    DC    A(ARG1)
       DC    A(ARG2)
       DC    A(ARG3)
```

ARG1, ARG2 et ARG3 étant des symboles d'adresses locales que l'assembleur détermine sous la forme P.D pour mise en place dans les positions correspondantes du tableau LST, à titre de constantes.

6.5 - Exécution d'instructions portant sur des chaînes de caractères

Ces instructions présentent une particularité importante que nous allons montrer sur l'instruction de mouvement de caractères MVC qui se présente sous la forme :


```
    MVC    ALPHA(L,X1),BETA(X2)
```

ou

```
    MVC    P1.D1(L,X1,B1 ),P2.D2(X2,B2*)
```

ALPHA est l'adresse de tête de la chaîne réceptrice de longueur L (octets) dans laquelle viennent les caractères logés à partir de l'adresse BETA (la longueur étant la même). Les adresses ALPHA et BETA peuvent être internes ou externes ; le point important est que les deux zones peuvent ne pas être chargées en mémoire centrale au moment où l'on décode l'instruction MVC, car ces zones peuvent chevaucher sur plusieurs pages successives.

Plusieurs possibilités existent pour régler ce problème ; la plus simple est de vérifier, compte tenu des adresses et de la longueur des chaînes, que les blocs désirés sont bien en mémoire centrale avant de lancer l'exécution. Sinon il faut provoquer le chargement des blocs manquants. L'exécution de l'instruction fait intervenir des zones successives dont l'adresse symbolique absolue s'établit en utilisant un compteur (pouvant jouer en index à partir de la position initiale). Une autre solution consistant à exécuter ce que l'on peut avant de bloquer obligerait à des modifications d'instructions plus délicates à mettre en oeuvre.

6.6 - Appel d'une variable externe

La variable est définie par une pseudo-instruction EXTRN qui fait réserver un double-mot à cette adresse

```
ALPHA        EXTRN        FCH,SYMB
```

FCH étant le n° de fichier dans lequel est définie cette variable et SYMB son adresse locale dans ce fichier.

ALPHA est le pseudonyme utilisé dans le programme actuel.

Le programmeur peut placer cette adresse externe dans un registre pointeur utilisable comme registre de base.

```
LP      3,ALPHA
```

provoque le chargement du n° de fichier FCH dans le registre 3 et celui de l'adresse locale SYMB dans le registre 3'. Ces deux quantités sont bien définies au moment de l'exécution d'un programme édité.

Le chargement de la quantité d'adresse [FCH,SYMB] dans le registre 7 peut alors se faire en utilisant l'instruction.

```
L       7,0.0(0,3*)
```

qui exécute

((3),0.0+(3')) ⟶ (7)

Si l'on écrit cette instruction sous la forme symbolique.

```
L       7,ALPHA
```

l'assembleur reconnaissant que ALPHA est une variable externe dont l'adresse est logée dans la paire 3-3' peut construire l'instruction

```
L       7,0.0(0,3*)
```

Il y a lieu d'indiquer à l'assembleur quels sont les paires utilisées comme registres de base et les variables externes dont les adresses sont actuellement logées dedans. Dans cette hypothèse, après

```
LP      3,ALPHA
```

on pourrait loger

```
USING   3,ALPHA
```

pour signaler que l'on veut l'utiliser comme registre de base.

Une autre solution est d'indiquer par USING les paires utilisables comme registres de base, laissant l'assembleur gérer leur utilisation. Celui-ci mettra en place les instructions LP de chargement, tiendra à jour une table du contenu de ces registres et effectuera un contrôle de non modification de ces registres par d'autres instructions que celles qu'il met en place. Le point important est qu'à chaque fois qu'il faut charger le registre de base, le double mot défini par EXTRN est là pour fournir les éléments voulus. L'assembleur peut évidemment optimiser l'utilisation des registres de base de façon à faire le moins de chargements possible.

6.7 - Appel d'un tableau externe

Le problème se présente de la même manière que dans le cas d'une variable externe, le rôle étant joué par le nom du tableau qui en désigne le premier octet.

On peut ainsi travailler par indexage dans un tableau externe TAB. Si l'on utilise la paire de registres de base 2-2', on placera l'instruction

```
LP      2,TAB
```

après avoir défini TAB par une pseudo-instruction EXTRN. Après chargement convenable du registre d'index 4' on pourra sortir le contenu du registre 7 dans cette position indexée par :

```
ST      7,0.0(4',2*)
```

qui provoque l'exécution de

(7) ⟶ ((2),0.0+(4')+(2'))

On aurait pu écrire également

```
ST      7,TAB(4')
```

qui aurait donné lieu à la création de la même instruction.

L'utilisation des registres de base comme registres pointeurs s'effectue de même. Ainsi, l'addition au registre 3' du contenu du 3ième mot de TAB s'effectuerait par

```
A       3',0.8(0,2*)
```

qui exécute

((2),0.8+(2'))+(3') ⟶ (3')

On pourrait également écrire

```
A       3',TAB+8
```

qui donnerait naissance à la même instruction.

L'indexage dans ces derniers cas est possible et n'introduit rien de nouveau.

Le point important dans ce domaine est que le travail sur des fichiers externes, dès lors que les registres de base sont chargés, s'effectue aussi facilement et aussi rapidement que si les variables et tableaux étaient internes.
Les constantes spéciales introduites par EXTRN établissent les liaisons entre les programmes d'une manière absolue.

6.8.- Appel d'un sous-programme externe

Ce travail a été vu lors de la description des instructions de branchement avec liaison. Le point d'entrée doit être placé dans une paire de registres de base ou être défini par rapport au contenu d'une telle paire. L'adresse de l'instruction suivante à exécuter se retrouve dans une paire qui sera utilisable comme paire de registres de base pour le retour.

Si LST est un tableau d'adresses d'arguments logé dans le programme appelant, (on a LST=$[n°prog1, P_1.D_1]$)et que le sous-programme appelé soit défini par

SP EXTRN n°prog2,$P_2.D_2$

on pourra appeler le sous-programme par :

```
LA      1,LST
LA      15,SP
BALR    14,15
```

qui provoque la création des instructions suivantes :

LA	1,$P_1.D_1$	n°prog1=(0) ⟶ (1),$P_1.D_1$ ⟶ (1')
LP	2,$P_9.D_9$	n°prog2 ⟶(2),$P_2.D_2$⟶ (2')
LA	15,0.0(0,2*)	(2) ⟶ (15),(2') ⟶ (15')
BALR	14,15	

l'instruction LP est mise en place par le programmeur ou par l'assembleur, suivant le mécanisme adopté pour l'assemblage.

Si SP est l'adresse : $[n°prog1, P_9.D_9]$ et contient $[n°prog2, P_2.D_2]$, la commande est bien passée au sous-programme externe ayant le pseudonyme interne SP.

L'appel de l'argument 1 d'adresse $[n°prog1, P_3.D_3]$ dans le registre 7 s'effectue au moyen des instructions.

L	5,0.0(0,1*)	((1),(1')=$P_3.D_3$ ⟶ (5)
puis L	7,0.0(5,1)	((1),(5))=((1),$P_3.D_3$) ⟶ (7)

Pour envoyer le contenu du registre 7' dans la position contenant l'argument 3, d'adresse $[n°prog1, P_5.D_5]$, on peut utiliser :

```
L       5,0.8(0,1*)     ((1),0.8+(1'))
                        =(n°prog1, P1.[D1+8])
                        =P5.D5 ⟶ (5)
puis ST 7',0.0(5,1)
```

qui exécute :

(7') ⟶ ((1),0.0+(5))=(n°progl,P_5.D_5)

6.9 - Travail en pile

On désigne ainsi les opérations s'effectuant sur un tableau externe par rapport à une position spéciale appelée sommet, dont l'adresse est dans une paire de registres jouant le rôle de pointeur. Ce pointeur peut évoluer au cours du travail. Il doit être possible de placer un élément au sommet de la pile et de monter ce sommet, de descendre le sommet et de lire l'élément situé à ce sommet, de lire ou de modifier un élément situé dans une position relative donnée par rapport au sommet.

La pile s'établit dans une réserve ayant comme adresse de départ : [n°fich1,P_1.D_1] ; dans le programme utilisateur, cette position aura le pseudonyme RES défini par

RES EXTRN n°fich1,P_1.D_1

On utilisera la paire 2-2' comme registre pointeur. Il faut d'abord faire le chargement du registre pointeur par

LP 2,RES n°fich1 ⟶ (2),P_1.D_1 ⟶ (2')

La mise en place d'un élément au sommet de pile et la montée de ce sommet peuvent se faire comme suit :

```
ST    7,0.0(0,2*)
LA    2,0.4(0,2*)
```

la première exécute

(7) ⟶ ((2),0.0+(2'))=((2),(2'))

tandis que la seconde majore le pointeur de 4 unités :

(2) ⟶ (2), 0.4+(2') ⟶ (2')

La descente du sommet de pile et la lecture de l'élément situé à ce sommet peuvent se faire au moyen des instructions suivantes :

```
S     2',P4
L     5,0.0(0,2*)
```

P4 étant une position contenant la quantité 4. On pourrait écrire également

S 2',=F'4' (2')-4 ⟶ (2')

l'instruction de chargement réalise :

((2),(2')) ⟶ (5)

Supposons maintenant que nous voulions charger le 3ième élément en dessous du sommet de la pile ; il suffit d'exécuter les instructions suivantes :

```
LA    4',0.12     12 ⟶ (4')
LNR   4',4'       -(4') ⟶ (4)=-12
L     8,0.0(4',2*)
```

Cette dernière instruction réalise :

((2),0.0+(4')+(2'))=((2),(2')-0.12) ⟶ (8)

ce qui réalise bien l'opération cherchée.

Il faudrait bien sûr vérifier à chaque évolution du sommet de pile que l'on ne déborde pas des limites de la réserve et émettre un message d'alerte en cas de débordement. Une interruption en cas de dépassement des limites sur le registre pointeur faciliterait beaucoup ce genre d'exploitation.

6.10 - Travail sur listes enchaînées

Une cellule de liste enchainée est un tableau auquel on peut accéder au moyen d'un pointeur, de préférence un registre pointeur. Certains éléments de cette cellule contiennent des informations caractéristiques tandis que certains autres contiennent l'adresse de la cellule suivante. Sous réserve de charger le même registre pointeur à partir de cette cellule, il est possible d'évoluer dans les cellules en suivant les enchaînements. Il est possible d'établir des enchaînements latéraux (reconnaissables grâce à des marques convenables) et de créer des structures ramifiées et même des graphes.

Pour fixer les idées nous allons considérer une cellule de 4 mots, les deux premiers pouvant contenir des données et les deux derniers formant pointeur à la cellule suivante. La paire de registres 2-2' est supposée contenir l'adresse de la cellule courante. Si l'on veut faire venir le contenu du premier mot dans le registre 7, il suffit d'employer l'instruction.

L 7,0.0(0,2*)

le chargement du deuxième mot se ferait par :

ST 5,0.4(0,2*)

le chargement du registre pointeur pour accéder à la cellule suivante se ferait simplement par :

LP 2,0.8(0,2*)

car cela correspond à :

((2),(2')+8) ⟶ (2)
((2),(2')+12) ⟶ (2')

Il serait ensuite possible de recommencer le même travail sur la cellule suivante.

Une structure très prometteuse pour le traitement de gros fichiers directs, sous réserve de pouvoir travailler en partage de temps pour compenser les temps d'appel successifs, réside dans le multienchaînement. Dans cette structure les cellules sont plus importantes, les positions données correspondent à des éléments caractérisés : données directes, pointeurs d'enchaînement d'articles définis, pointeurs de retour, pointeurs à des fiches de liaison. La connaissance de la structure de la cellule permet d'organiser le travail d'exploration suivant une caractéristique donnée et le travail de détermination d'une autre caractéristique relative à la cellule considérée.

A titre d'exemple, nous allons considérer un fichier de personnel équipé suivant cette structure. Pour simplifier les choses, sur la Fig. 3, nous supposerons que les pointeurs (qui correspondent à des doubles-mots) sont représentés par une case unique. Des tableaux indexés, des tables consultables par exploration, des arborescences permettent d'accéder à des pointeurs correspondant à des caractéristiques précises recherchées. A partir de là, l'exploration de l'enchaînement permet de passer en revue les cellules des individus possédant cette même caractéristique particulière. Sur chaque cellule, il est possible de connaître d'autres caractéristiques en utilisant les pointeurs de retour correspondant à ces autres caractéristiques, pointeurs qui ramènent à des positions contenant ces caractéristiques en clair, suivant un code convenu, donc exploitable automatiquement. Les caractéristiques uniques : situation de famille, classement hiérarchique, actuel, etc.. peuvent correspondre à des enchaînements directs ; les caractéristiques multiples, telles par exemple les langues connues de l'individu, demandent l'utilisation de fiches de liaison sur lesquelles se font les enchaînements.

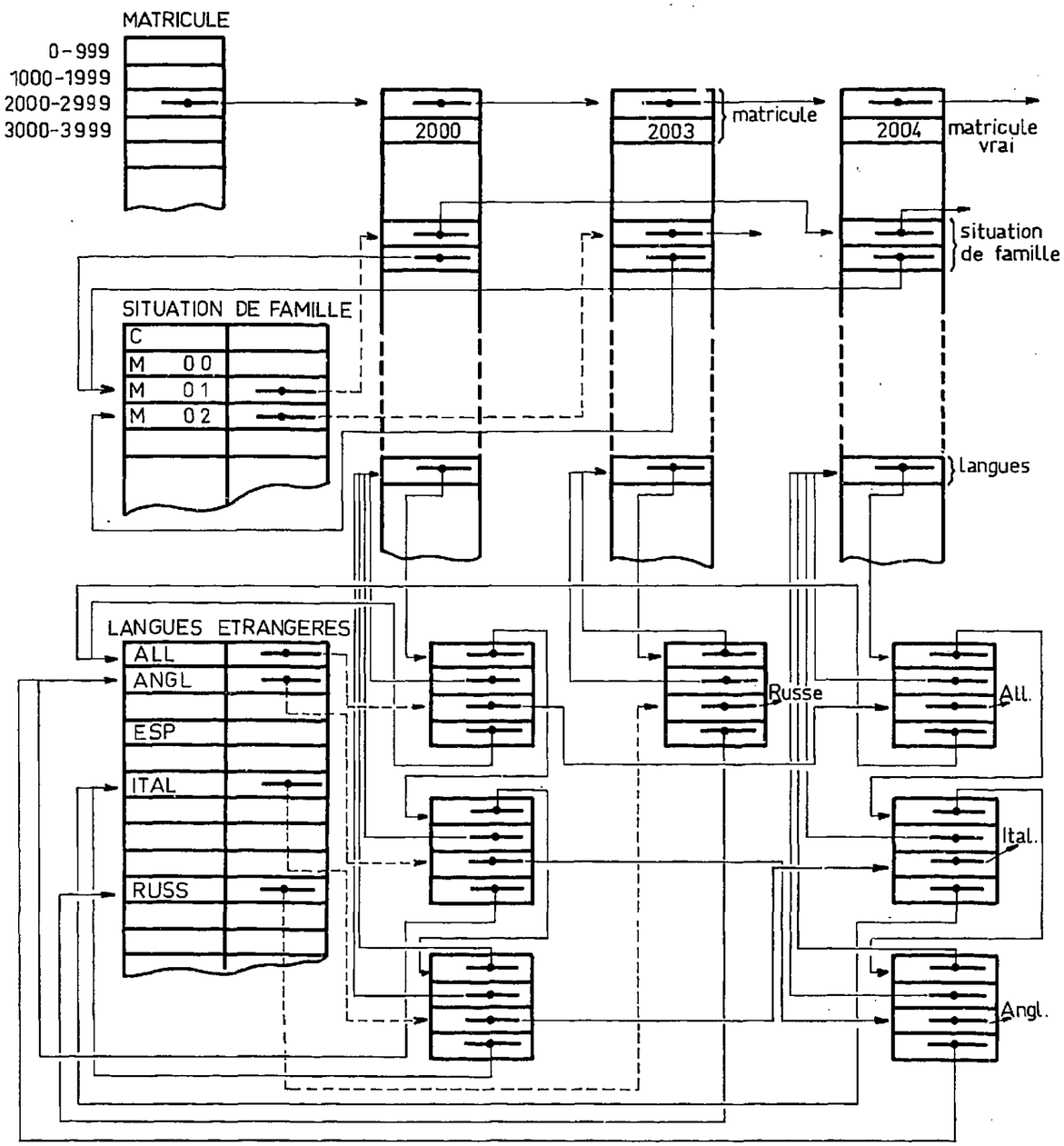


Fig 3 - Listes multienchaînées.

Sur la Fig.3, des fiches individuelles sont enchaînées par numéros matricules. La table MATRICULE est consultable par indexage car elle donne le pointeur au début de chaque tranche de 1000. La tranche 2000-2999 est indiquée sur cette figure. La table SITUATION DE FAMILLE doit être explorée pour chercher la caractéristique correspondante (célibataire, marié sans enfant, marié avec 1 enfant, etc...) ; en correspondance se trouve le pointeur de la chaîne considérée.
La rubrique SITUATION DE FAMILLE est dans une position bien déterminée dans chaque fiche et comprend un pointeur d'enchaînement et un pointeur de retour permettant de lire la désignation symbolique correspondante.
La rubrique LANGUES est dans une position fixée de chaque fiche mais ne contient qu'une position logeant un pointeur d'enchaînement de fiches de liaison dont le nombre dépend des langues connues. La table LANGUES ETRANGERES est consultable par exploration et permet de déterminer le pointeur d'enchaînement associé à chaque langue. Les fiches de liaison sont formées de deux éléments : le premier sert à enchaîner les fiches d'un même individu et à fournir le pointeur de retour à la fiche de cet individu, le deuxième contient le pointeur d'enchaînement de la caractéristique correspondante et le pointeur de retour à cette caractéristique particulière.

7 - SUPERVISION

Les travaux doivent être dirigés par un superviseur logé dans un certain nombre de blocs physiques ayant un statut particulier. Les blocs du superviseur sont indiqués dans le registre associatif dans les cellules correspondant aux blocs physiques occupés par celui-ci. L'indicateur BSV (voir Fig.2) permet de sélectionner les blocs du superviseur et les autres, ou les autres seulement, suivant que l'on est en régime "superviseur" ou en régime "utilisation". Le passage du deuxième régime au premier a lieu lorsque l'on exécute un appel superviseur (ou

une interruption se traduisant par un tel appel). L'intérêt de BSV est de pouvoir passer d'un régime à l'autre sans modifier RASR, ce qui permet de reprendre instantanément la tâche en cours.

7.1 - Affectation d'une tâche

Lors de toute demande de travail le superviseur attribue, s'il le peut, un numéro de tâche, ouvre une fiche de tâche relative à ce travail et affecte un bloc de service qu'il suivra pendant toute l'exécution. D'autres blocs de service seront affectés en cas de besoin au cours du travail.

Le demandes et les échanges entre l'utilisateur et le superviseur se font, généralement, via une machine à écrire branchée sur un canal. Le tampon du canal contient les messages à échanger ; pendant la frappe l'ordinateur est déchargé de la tâche de transmission et travaille sur autre chose.

Au début d'un travail, des informations d'identification sont échangées et l'ordinateur indique à l'utilisateur la composition des fichiers qui lui appartiennent, ainsi que le temps de travail dont il dispose (dans une organisation à affectation préalable de temps de travail). Ceci permet d'ailleurs de mettre dans la fiche de tâche les renseignements qui seront utiles en cours d'exécution et qui sont approvisionnés en consultant le répertoire général des utilisateurs dont la tête est en mémoire centrale et le reste en mémoire de stockage (tout au moins si ce répertoire général est volumineux).

Dès que le superviseur a lancé ses ordres d'approvisionnement, il revient à la tâche qui était en cours d'exécution et qui a été interrompue. Ceci se fait en rebasculant BSV.

7.2 - Changement de tâche

Si un blocage se produit, ou si une interruption par l'horloge interne indique que le temps alloué pour cette tournée est écoulé, le superviseur provoque la mise en garde de l'état machine (registres de travail, registres associatifs et indicateurs internes) et rétablit l'état machine pour la tâche à laquelle il va passer la commande.

Cette tâche qui prend la commande est déterminée par consultation de la file d'attente. Le superviseur consulte en même temps cette liste pour savoir quelle sera la tâche qui viendra après, de façon à lancer un ordre d'approvisionnement des blocs utiles pour cette tâche, si cela est possible.

Lors de la mise en garde de RASR dans le tableau de service, celui-ci constitue la table d'affectation mémoire de la tâche. Lors de la rentrée en service, cette table sert à recharger RASR. Un contrôle, effectué avec la table d'affectation mémoire du superviseur, permet de vérifier que tous les blocs inscrits sont bien présents dans la mémoire physique, aux endroits indiqués.

Lors du rétablissement de l'état machine on remet en place, en particulier, le bloc (ou les blocs) de service connu par la fiche de tâche, le registre pointeur qui permet d'y accéder et les registres O et O', qui permettent de reprendre la tâche où elle avait été interrompue.

La file d'attente (qui est en réalité la table de gestion des tâches, suivant les priorités retenues) peut introduire des subdivisions plus fines que les tâches, si les programmes de ces tâches fournissent des renseignements permettant un fonctionnement asynchrone. Dans le

cas considéré ici, cela permet de donner à chaque tâche une meilleure part du temps qui lui est attribué à chaque fraction d'exécution ; dans le cas d'ordinateurs travaillant en multitraitement cela serait encore plus important.

Quand un bloc non chargé est appelé, la commande est repassée au superviseur qui consulte RAMT pour savoir quel est le bloc appelé non chargé puis, après consultation de la fiche de la tâche ou de la liste des programmes généraux, décide d'envoyer un message indiquant un appel non autorisé ou d'approvisionner le bloc demandé. Dans ce dernier cas, il consulte la table d'affectation de la mémoire pour voir s'il y a de la place libre et, dans ce cas, affecter un bloc libre au bloc demandé. Ceci se traduit par l'inscription de ce bloc (avec indication de non chargement) dans la table d'affectation de la mémoire et dans la position correspondante du registre associatif, par un lancement d'instruction d'entrée, après consultation de la fiche consacrée à la tâche ou de la liste des programmes généraux. Ensuite, la commande est passée à la tâche suivante, comme précédemment.

S'il n'y a pas de bloc libre, deux possibilités se présentent :

1°) - on sélectionne un bloc pour le renvoyer en mémoire auxiliaire, suivant des critères à fixer par le système (algorithme de gestion). Ceci s'accompagne d'une mise à jour de la fiche de tâche relative au bloc renvoyé, et d'une mise à jour de la table d'affectation de la mémoire et de RASR. L'ordre de sortie en mémoire auxiliaire du bloc renvoyé est lancé et l'ordre d'appel du bloc demandé est mis en réserve dans la file d'attente.

2°) - on indique le bloc à approvisionner dans la file d'attente. Lors de l'exécution de la tâche précédant la reprise de la tâche relative à

cet appel, l'approvisionnement sera lancé, éventuellement en faisant de la place comme indiqué ci-dessus. Compte tenu des temps de chargement la prévision des lancements peut être à plus longue échéance.

Il y a lieu de remarquer que des digits de contrôle peuvent être positionnés dans la table d'affectation mémoire, de façon à indiquer qu'un bloc est appelé mais non encore chargé. A la fin du chargement d'un bloc une interruption permet la mise à jour du digit de contrôle dans la table d'affectation mémoire et indique ce fait dans la fiche de tâche correspondante (qui appartient au superviseur).

Lorsque le travail relatif à une tâche est achevé, la fiche de tâche permet la mise à jour du répertoire général, compte tenu des évolutions subies ; ensuite, elle est mise à zéro ainsi que les blocs de service et ces éléments sont rendus au superviseur pour un autre utilisateur. Des marques mises en place dans RASR, recopiées lors des mises en garde, permettent de piloter les appels mémoire, en indiquant si seule l'écriture ou la lecture sont autorisées sur certains blocs et d'indiquer si un bloc n'a travaillé qu'en lecture (ce qui évite d'avoir à le renvoyer en mémoire de stockage, puisqu'il n'a pas été modifié).

Lorsqu'il n'y a plus de travail à faire en partage de temps, le superviseur doit donner la commande à un volant de programmes à exécuter par trains, de façon à assurer le travail de l'ordinateur.

Un problème important est de déterminer la taille de la mémoire centrale et son mode de subdivision en blocs physiques. Il faut qu'il y ait un nombre suffisant de blocs si l'on veut pouvoir garder les éléments de plusieurs tâches simultanément mais le nombre de dispositifs d'accès et de cellules de la mémoire associative augmente et cela agit de façon importante sur le prix de revient.

En principe, à un instant donné, seules les pages intéressantes ont besoin d'être en mémoire centrale, ce qui est avantageux, en particulier dans le cas de programmes de grande taille. Cela amène à une politique d'assemblage tendant à ne pas placer de parties de programmes correspondant à des boucles à cheval sur plusieurs pages, mais à les mettre entièrement sur une page (ou plusieurs), quitte à ne pas remplir une page donnée que l'on quitte par un branchement.

L'un des blocs correspondant au superviseur peut être de taille plus importante si les appels que l'on doit y faire s'effectuent au moyen de registres de base utilisés en pointeurs. En effet, dans ce cas, l'adresse située dans le deuxième registre peut s'établir dans la zone déplacement et se continuer en avant de la zone page, ce qui laisse toute latitude d'adressage. Ce bloc particularisé est surtout utile pour loger les listes enchaînées servant à la gestion générale, listes qui doivent rester en permanence dans cette zone.

- UTILISATION DE PROGRAMMES COMMUNS - RECURSIVITE

Les programmes communs doivent être formés de deux parties : l'une, complètement fixe, est constituée par les instructions proprement dites du programme ; l'autre, variable suivant la tâche qui l'utilise, est logée dans le tableau de service de cette tâche et appelée par l'intermédiaire d'un registre pointeur. Ceci veut dire que les blocs communs, une fois chargés, apparaissent simultanément dans les tables d'affectation mémoire de toutes les tâches qui les utilisent, tandis que les parties variables sont dans les tableaux de service de ces tâches. Le superviseur suit ces programmes communs et ne les renvoie en mémoire auxiliaire qu'en dernière extrêmité.

La même politique est à suivre pour les programmes qui doivent travailler en récursivité directe ou indirecte et, non plus seulement en concurrence interne (appelée aussi réentrance) : les éléments variables au cours des différentes exécutions doivent être logés dans le tableau de service de la tâche et être appelés par l'intermédiaire du registre pointeur correspondant.Ces tableaux doivent être logés en pile pour permettre d'en établir un par appel.

Le superviseur peut suivre la montée du sommet de pile dans le tableau de service et en introduire un autre quand le premier est plein, ou rétablir le tableau de service précédent quand le sommet redescend.

Il y a lieu de remarquer que le logement en pile des tableaux d'éléments variables d'un programme (mémoires de travail, tableaux dynamiques instructions modifiables, etc..) n'introduit pas de difficulté grâce au travail par l'intermédiaire de registres pointeurs, dès lors que le programme a été établi en conséquence. Cela n'introduit pas non plus de supplément appréciable de la durée d'exécution tout en rendant automatiquement récursifs les programmes correspondants.

A chaque entrée dans un programme le registre pointeur de travail est mis à jour en utilisant le pointeur de la pile et la taille connue du tableau ; le pointeur de pile est ensuite mis à jour. Lors de la sortie du programme, les opérations inverses sont effectuées. A titre d'exemple, nous allons voir comment cela se passe avec deux programmes dont le deuxième appelé par le premier, rappelle celui-ci.

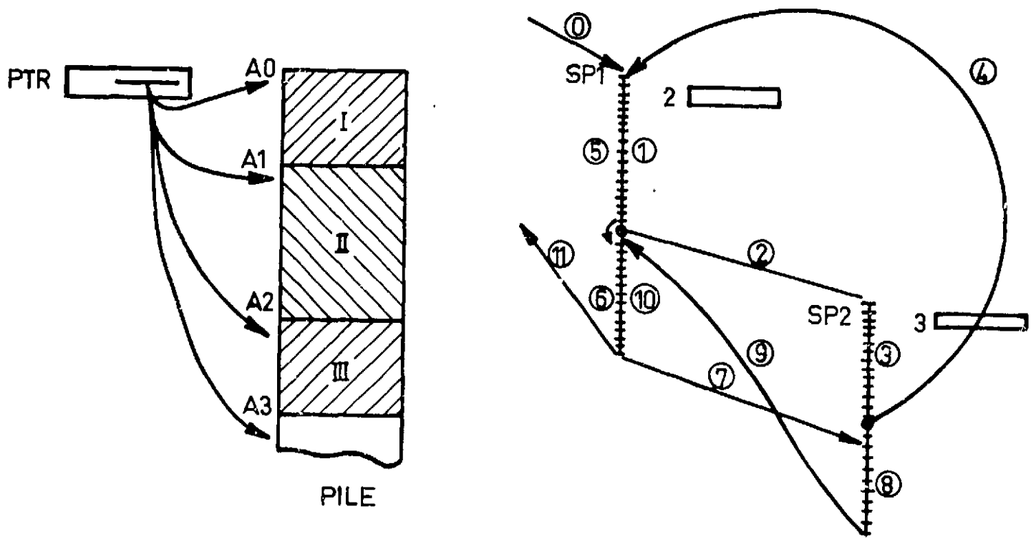


Fig.4 - Exécution d'un programme récursif SP1 rappelé par un programme SP2 qu'il appelle lui-même. L'exécution suit les numéros indiqués sur les programmes et sur les flèches de branchement.

On suppose que les registres pointeurs utilisés par les deux programmes SP1 et SP2 sont respectivement les paires 2-2' et 3-3'.

A l'entrée dans un programme les registres sont mis en garde dans le tableau du programme appelant, puis le contenu du pointeur de réserve PTR est majoré de la taille du tableau et cette valeur est chargée dans le pointeur de travail.

A la sortie, les registres sont rétablis (y compris le pointeur du programme appelant), le pointeur de réserve PTR est mis à jour compte tenu de la taille du tableau que l'on quitte.

A l'arrivée dans SP1, (PTR)=A0 ; après l'entrée (PTR)=A1 et le registre pointeur 2 contient également A1. Le travail s'effectue en utilisant le tableau I. Lors de l'appel de SP2, le contenu du registre 2, soit A1, est mis en garde dans le tableau 1, (PTR)=A2 ainsi que le contenu du registre pointeur 3. Le travail s'effectue en utilisant le tableau II. Lors du deuxième appel de SP1 (chemin ④), le registre 3 est mis en garde dans II, (PTR)=A3 ainsi que le contenu du registre 2. Le travail s'effectue en utilisant le tableau III jusqu'à achèvement de SP1. A la fin, le registre 3 est rechargé de A2 par rétablissement des registres et (PTR)=A2 par remontée de pile. La partie ⑧ du programme SP2 s'effectue en utilisant le tableau II. A la fin de SP2, le registre 2 est rechargé de A1, (PTR)=A1 par remontée de pile. La partie ⑩ de SP1 s'effectue en utilisant le tableau 1. A la fin de SP1, les registres sont rétablis, (PTR)=A0 par remontée de pile et l'on retourne au programme appelant.

9 - REVUE DES CARACTERISTIQUES D'ENSEMBLE DE LA STRUCTURE PROPOSEE

Nous allons rappeler les points principaux relatifs à cette structure d'ordinateur :

1°) - Elle peut permette le travail en partage de temps d'une façon souple, sans compliquer inutilement les appels de la mémoire centrale.

2°) - Elle assure, avec le minimum de travail la gestion des affectations mémoire, des chargements et des liaisons.

3°) - Le chargement peut être entièrement dynamique ; seuls les blocs utiles sont à charger à tout instant, ce qui permet une utilisation optimale de la mémoire.

4°) - Le travail avec des tableaux et des listes dynamiques est facile.

5°) - La protection mémoire est assurée automatiquement.

6°) - Le travail simultané avec des programmes communs ou récursifs est facile.

7°) - La gestion d'ensemble d'un très grand système de programmes est facilitée, quels que soient les supports utilisés pour les mémoires de stockage. Les appels sont toujours nominatifs, le répertoire général fournissant au superviseur les renseignements voulus pour élaborer ses ordres.

8°) - Le système doit évidemment comprendre les compilateurs correspondant aux différents langages utilisables, qui feront l'assemblage dans le cadre de cette structure, en consultant le répertoire des fichiers pour savoir quel est le premier fichier disponible et affecter son numéro au programme correspondant. Un compilateur peut d'ailleurs indiquer qu'un programme est trop long et demander son sectionnement.L'édition des programmes est facile à réaliser.

Dans ce que nous avons vu plus haut, deux mémoires intervenaient : la mémoire centrale rapide appelée par l'intermédiaire des registres associatifs RASR et la mémoire de stockage à accès sélectif d'accès plus long.

En fait, il paraît commode, lors de l'appel d'un bloc d'un fichier d'amener le fichier correspondant sur une mémoire auxiliaire (tambour rapide, par exemple) de façon que les appels ultérieurs à ce fichier puissent être exécutés rapidement. Il est d'ailleurs possible d'utiliser une mémoire associative du même type que RASR pour connaître l'adresse tambour d'un fichier (et non d'un bloc), les adresses des blocs s'en déduisant automatiquement. Si l'on appelle RASL cette mémoire associative les opérations d'appel deviendraient les suivantes :

1°) - le bloc appelé se trouve en mémoire centrale ; une ligne est sélectionnée par RASR.

2°) - sinon, si le bloc se trouve sur le tambour, une ligne de RASL est sélectionnée.

3°) - sinon, il faut consulter le répertoire pour lancer l'appel dans la mémoire de stockage.

Cette disposition demande des instructions de chargement et de rangement dans RASL. La protection du tambour est assurée du fait que RASL ne contient que les n^{os} des fichiers autorisés pour la tâche en cours.

Il ne semble pas nécessaire d'avoir d'autres types de mémoire sauf éventuellement des mémoires d'archivage (bandes magnétiques, par exemple). Le tambour peut être avantageusement remplacé par une mémoire à tores moins rapide que la mémoire centrale mais de grande taille.(type LCS, par exemple).

10 - RESEAUX D'ORDINATEURS

La structure précédente n'est rentable que lorsque l'on peut effectuer plusieurs travaux simultanément de façon à faire travailler l'unité centrale d'une manière continue. C'est la raison pour laquelle il faut avoir un volant de travaux de routine venant meubler les trous du traitement en dialogue. Il y a également possibilité de gérer des ordinateurs périphériques faisant du multitraitement d'une manière systématique ou occasionnellement.

Ceci nous amène à faire quelques remarques sur la conception des centres ou des réseaux de traitement de l'information. Un argument

souvent mis en avant, et d'ailleurs valable dans un certain contexte, est qu'il y a intérêt à utiliser les plus groses machines possibles parce que le prix de revient de l'instruction élémentaire est bien plus faible que pour des machines petites ou moyennes. L'exécution de travaux de routine ou de travaux volumineux fréquents répond bien à ce genre de demande et justifie une très grosse installation. Il n'en est plus de même lorsqu'une priorité absolue de traitement est demandée, comme par exemple dans la commande de processus industriels ou expérimentaux. Bien que les interruptions de programme prioritaires puissent permettre de traiter certains travaux de ce type, il arrive un moment où le temps de réponse possible devient plus grand qu'il n'est admissible lorsque le nombre de tels équipements accrochés à l'ordinateur augmente. D'autres travaux prioritaires peuvent justifier un ordinateur séparé de taille plus modeste pour des raisons de facilité d'utilisation, bien qu'un partage de temps correct puisse permettre d'éviter une telle solution.

Compte tenu des remarques précédentes, il nous semble que l'idéal est d'affecter un ordinateur de taille adaptée à chaque utilisateur, ou groupe d'utilisateurs, demandant une priorité absolue pour leurs travaux. L'adaptation peut amener à des ordinateurs de grande taille si les utilisateurs font du travail de routine.

Il y a intérêt à ce que ces ordinateurs soient interconnectés pour former un réseau. Ceci peut se faire de différentes façons :

1°) - Un ordinateur donné peut être connecté à un ou deux ordinateurs voisins susceptibles de prendre ses tâches en charge en cas de défaillance, ce qui augmente la fiabilité d'ensemble du système.

2°) - Chaque ordinateur peut être connecté à un ordinateur de grande taille jouant le rôle de chef d'orchestre.L'ordinateur dessert

ses tâche prioritaire normalement et signale au système sa disponibilité dès qu'il est libre. Le superviseur du système organise alors la distribution des travaux en fonction des équipements disponibles de façon à optimiser le travail d'ensemble. Cette structure présente l'avantage de pouvoir mettre, éventuellement, les ressources du système à la disposition de chacun puisque les demandes de traitement peuvent être adressées par tous au superviseur central. L'utilisation groupée permet de ne pas trop lésiner sur l'équipement d'un système à priorité absolue mais travaillant relativement rarement, puisque le matériel peut être utilisé par le système.

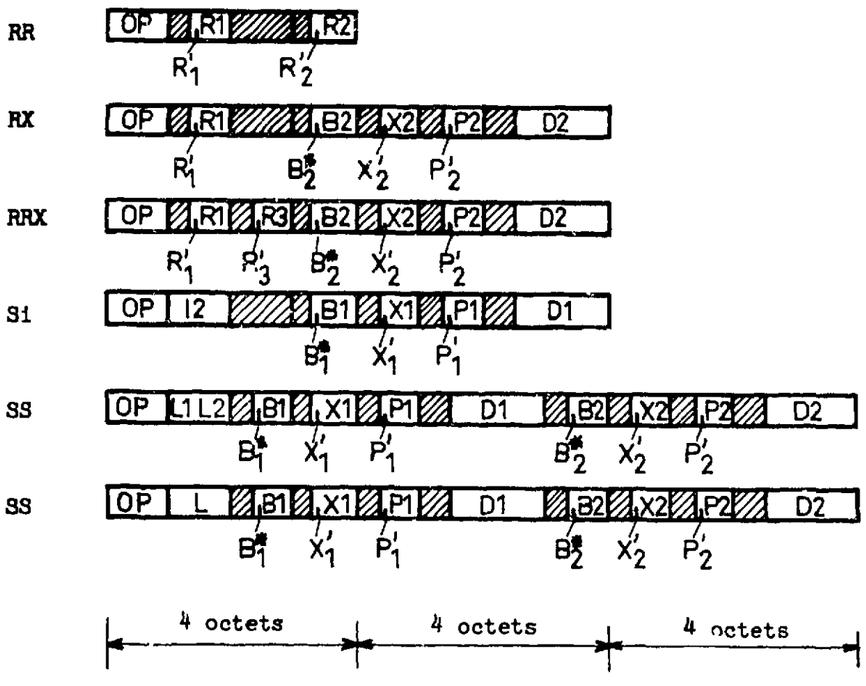
Les remarques précédentes indiquent une philosophie de l'emploi des ordinateurs qui nous paraît intéressante en même temps qu'elles indiquent la place d'une structure telle que celle que nous avons décrite plus haut . Il semble intéressant de rechercher le développement des traitements automatiques plutôt dans l'organisation et dans l'utilisation des travaux parallèles plutôt que dans la recherche des vitesses absolues d'exécution. N'oublions cependant pas que les méthodes envisagées sont basées sur des ordinateurs du type Von Neumann, c'est à dire faisant du traitement séquentiel en utilisant des programmes enregistrés. Des machines faisant un travail typiquement parallèle sont à l'étude et pourront donner des résultats importants dans l'avenir. Des développements intéressants nous semblent cependant réalisables, assez facilement, avec les structures actuelles, dans les voies que nous avons esquissées dans cet exposé.

Manuscrit reçu le 21 juin 1968

NOTE ADDITIONNELLE

Les formats d'instructions indiqués dans le texte peuvent compliquer l'appel de celles-ci. Il peut être plus facile de prendre trois formats de 1, 2 et 3 mots et de faire la décomposition suivante qui autorise un découpage en mots tout en donnant une lecture facile des instructions hexadécimales ; ceci compense la place perdue dans les instructions. Le déplacement est toujours indiqué par 12 digits binaires alors que le numéro de page l'est par un groupe de 8 digits permettant des fichiers de 256 pages de 1024 mots, le nombre total de fichiers possibles restant le même (supérieur à 4 milliards).

Ces nouveaux formats d'instructions sont les suivants :



Note :les parties hachurées sont à zéro et peuvent être utilisées pour des extensions éventuelles.

- B I B L I O G R A P H I E -

- S. Gill — Introduction to Time - Sharing, in Peter Wegner : Introduction to System Programming Academic Prss, 1964

- William H. Desmonde Real-Time Data - Processing Systems : introductory concepts Prentice Hall, Inc, 1964

- F.J. Corbato' , M. Merwin - Daggett, R.C. Daley An experimental time - sharing system in Proc. 1962 Spring Joint Computer Conference pp 335-344

- Peter Wegner Priority assignment in a time - shared Computer Installation Comm. ACM 5 Feb. 1962 pp 95 et 114

- M. Landis, A. Manos, L.R. Turner Initial experience with an operating multiprogramming system Comm. ACM 5 May 1962 pp 282 - 286

- R.V. Head — Real - time programming specifications Comm. ACM 6 July 1963 pp 376 - 383

- J.B. Dennis — A multiuser computation facility for education and research Comm. ACM 7 Sept. 1964 pp 521 - 529

- E.T. Irons — A rapid turnaround multiprogramming system Comm. ACM 8 March 1965 pp 152 - 157

- R.M. Brown An experimental study of an on - line Man - Computer system
IEEE Trans on E.C. - EC - 14 Feb. 1965 pp 82-85

- I.C. Pyle Data input by question and answer
Comm. ACM 8 April 1965 pp 223 - 226

- J.B. Dennis Segmentation and the Design of Multiprogrammed Computer System
Jl. A.C.M. 12 October 1965 pp 589 - 602

- James Martin Programming Real-Time Computer System
Prentice-hall, 1965

- AFIRO 1966 Utilisation des ordinateurs à distance en temps réel et en temps partagé
Dunod, 1967

- W.J. Karplus On-line Computing, Time-Shared Man-Computer system
MC Graw-Hill, 1967

- J.R. Ziegler Time-Sharing Data Processing Systems
Prentice-hall, 1967

- J. Bertin, M. Ritout, J.C. Rougier
L'exploitation Partagée des Calculateurs
Dunod, 1967

- D.C. Evans, J.Y. Leclerc
Address mapping and the control of access in an interactive computer
AFIPS Conference Proceedings 30 1967 SJCC pp 23 - 30
Academic Press, 1967

FIN