HN/cg
1.8.1976

LAB II-CO/74-2 ler décembre 1974

(SPS/AOP/CO-Note/77-3)
18.1.1977

ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE

LE SYSTEME NODAL POUR LE SPS - 1974

M.C. Crowley-Milling, J.T. Hyman et G.C. Shering

RESUME

Ce rapport regroupe des informations sur le système de logiciel NODAL. Il contient un manuel de programmation et fournit des renseignements sur la rédaction de modules en code machine pour NODAL. De plus, il présente un aperçu du système, qui devrait être utile pour ceux qui étudient comment utiliser NODAL en vue de la commande des différentes parties de l'accélérateur.

TABLE DES MATIERES

(principales subdivisions)

ANNEXES

TABLE DES MATIERES

(présentation détaillée)

12. Organisation du système NODAL

12.1 Système simple utilisant uniquement la mémoire centrale

12.2 Système pour un ordinateur de pupitre

12.3 Le système pour un ordinateur polyvalent

12.4 Le système pour un ordinateur affecté à une zone d'expérimentation

1. Introduction

Ce rapport décrit le système NODAL utilisé pour la commande du SPS, dans l'état atteint à la fin de 1974. Le document traite de deux grandes questions: en premier lieu, le langage NODAL; en second lieu, le système NODAL.

Le langage NODAL est un langage de programmation évolué, présentant des caractéristiques particulières en vue de son emploi pour la commande en temps réel, les applications faisant intervenir plusieurs ordinateurs, et les applications de type dialogique. Il est basé sur les langages FOCAL et SNOBOL-4, avec une certaine influence du BASIC.

Le système NODAL correspond à la technique de mise en oeuvre et d'emploi du langage NODAL dans le cadre de l'équipement et du logiciel fondamentaux du système de commande par ordinateur du SPS. L'équipement comprend 24 ordinateurs NORD-10 reliés entre eux par un système de transmission de messages en mode série, comprenant un autre ordinateur NORD-10. Le logiciel de base représente une version étendue du système d'exploitation SINTRAN II des ordinateurs NORD-10, appelée SYNTRON, comportant un système de gestion de fichiers sur plusieurs ordinateurs.

Toutes les ressources décrites dans ce rapport ne seront pas introduites dans tous les ordinateurs. En particulier les fonctions définies de NODAL (chapitre 8) et les ressources pour la manipulation de chaînes de caractères (chapitre 9) ne seront disponibles que sur l'ordinateur du pupitre, et les ressources évoluées de traitement par chaînage pour l'adaptation de configurations ne seront disponsibles que sur l'ordinateur de service.

Le chapitre 2 décrit les ressources fondamentales de NODAL pouvant être utilisées sur tout terminal dialogique relié à un ordinateur quelconque. Il est recommandé au débutant de lire attentivement ce chapitre puis de se rendre à un terminal du système de traitement en temps partagé afin de tenter de rédiger quelques programmes simples.

2. Eléments fondamentaux de la programmation en NODAL

2.1 Ordres simples TYPE et SET

NODAL est un système interprétatif dialogique, et il est donc utilisé pour l'écriture directe en ligne de suites de caractères sur le dispositif de contrôle qui peut être soit un télétype soit une console de visualisation (affichage sur tube à rayons cathodiques et clavier). Le plus petit élément utile de NODAL est l'ordre. Par exemple:

```
TYPE BCT(3)
```

est un ordre qui peut commander l'écriture sur l'écran de la valeur de la troisième lecture de l'intensité du faisceau en circulation.

```
SET INJPHS=12
```

est un autre ordre qui établira la phase HF à l'injection à 12 degrés. Les ordres se composent d'un mot-clé de commande, TYPE et SET pour les exemples ci-dessus, suivi par le corps de l'instruction dont la syntaxe varie d'une instruction à l'autre.

Un résumé de tous les ordres est présenté à l'Annexe 1.

2.2 Nombres, variables et expressions

La syntaxe du corps de l'instruction comprend pour l'essentiel des nombres et des variables, souvent associés dans des expressions.

2.2.1 Nombres

En NODAL, un nombre est normalement formé d'une suite ininterrompue de chiffres contenant un seul point décimal. Voici des exemples de nombres:

1 .1 6.25

Les nombres sont stockés avec une précision d'environ neuf chiffres décimaux, bien que les imprimés soient arrondis à six chiffres, sauf demande contraire.

En plus de la forme précédente, quelques autres cas particuliers de représentation de nombre sont acceptés par le système. A titre d'exemple citons l'écriture:

1.3E-4

qui signifie 1.3 fois dix à la puissance moins quatre. On peut stocker des nombres avec des exposants decimaux allant jusqu'à ± 4000, mais seuls des exposants d'au plus ± 2400 peuvent être convertis en chaînes de caractères.

2.2.2 Entièrs en Octal

Les entièrs compris entre -32768 et 32767 peuvent être représentés à l'aide d'une suite de chiffres écrits en octal (0-7), et prècédés du caractère " ["; par exemple:

[177777

répresente l'entier -1.

2.2.3 Entièrs écrits en alphanumérique RADIX 36

Des entiers compris entre -32768 et 32767 peuvent également être representés sous la forme d'une chaîne de caractères RADIX 36 comprenant les chiffres 0-9 (ayant leurs valeurs normales) et les lettres A à Z (représentant les entiers de 10 à 35). Si une chaîne est précédée du caractère " # ", par exemple:

#ABC #A1 #123

elle est considérée par NODAL comme représentant un entièr du type RADIX 36. Dans la chaîne de caractères, chaque position vaut 36 fois la position située immédiatement à droite, au lieu de 10 fois dans le système décimal et de 8 fois, en octal. L'algorithme utilisé fournit un entier positif modulo 2 à la puissance 16, que NODAL considère ensuite comme un entier affecté d'un signe.

2.2.4 Variables

Un nom de variable est formé d'une suite d'au plus six caractères (aucun espacement n'est permis). Le jeu de caractères comprend les lettres A-Z, les chiffres 1-9, avec en plus les deux points (:) et le point (.). Un nom doit commencer par une lettre. Voici des exemples de noms:

A ABC PS.5 VB:INJ

Les noms permettent de distinguer deux types de variables : les variables du programme et les variables du système. Les variables du programme peuvent être créées automatiquement par les ordres SET et ASK. Par exemple, l'ordre:

SET A=1

crée une variable du programme, A, si celle-ci n'existait pas déjà. Les variables du programme doivent obligatoirement contenir plus de deux caractères lorsqu'elles contiennent un point.

Les variables du système doivent contenir au moins trois caractères. Elles peuvent contenir deux points, à des fins mnémoniques. Ainsi:

A B1 I.MAX SIGN.1

sont des variables du programme alors que:

ABC PlA INJ:VB PS:1

sont des noms de variables du système, et de ce fait elles ne peuvent jamais être créées par les ordres SET et ASK.

2.2.5 Expressions

Les expressions sont des combinaisons de nombres, de variables et d'opérateurs mathématiques qui peuvent être évaluées par le programme d'interprétation, afin de fournir un résultat numérique. Par exemple, l'expression:

10 ↑ 3 * 3/10 + 21 - 2

peut être évaluée pour donner le nombre 319. Les opérations arithmétiques sont effectuées de gauche à droite, toutefois l'exponentiation (↑) est effectuée en premier, en étant suivie de la multiplication (*), de la division (/), de la soustraction (-) et de l'addition (+). Ainsi l'évaluation de 6 + 6 * 2 fournit 18, car la multiplication est effectuée avant l'addition. Des parenthèses peuvent être utilisées pour modifier l'ordre des opérations; ainsi l'expression (6 + 6) * 2 donnera 24. Les expressions peuvent contenir des nombres, des variables du programme et des variables du système, comme:

```
TYPE 2*K*BCT(3)
```

ou

```
SET INJPHS = A+2*B
```

Il convient de noter ici une différence par rapport au FORTRAN. EN NODAL la multiplication a une priorité plus élevée que la division, alors que ces priorités sont égales en FORTRAN. Ainsi:

```
1/2*PIE*I = 1/(2*PIE*I)
```

et

```
A/B*C = A/(B*C) et non pas (A/B)*C
```

2.3 Lignes et ordre FOR

La subdivision la plus importante d'un programme en NODAL est la ligne de texte. NODAL n'intervient pas tant que l'on n'a pas appuyé sur la touche de retour de chariot, <CR> , sur le clavier. Ce n'est qu'ensuite que la ligne est lue et que les ordres sont exécutés. Une ligne peut comprendre plusieurs instructions séparées par des points-virgules ; exemple :

```
SET A=1; SET B=2; SET INJPHS = 2*A+5*B
```

2.3.1 L'ordre FOR

L'ordre FOR utilise la fin de la ligne pour définir son champ d'action; par exemple l'instruction :

```
FOR I=10,20; SET VACVLV(I)=1
```

provoquera l'ouverture des vannes de vide 10 à 20, inclusivement. Des variables du système peuvent également être utilisées pour les ordres FOR, par exemple l'instruction :

```
FOR INJPHS = 6,18 ; TYPE INJPHS, BCT (3) !
```

provoquera l'affichage d'un tableau contenant les valeurs de la phase HF à l'injection et de l'intensité du faisceau en circulation. Le caractère! provoque l'impression d'une nouvelle ligne. L'ordre FOR comprend en général trois expressions, par exemple A, B, C, et s'écrit :

```
FOR I = A, B, C ; ...
```

où I prend initialement la valeur A, puis est incrémenté de B pour atteindre C. Si B ne figure pas, il est supposé égal à l'unité. Si on le désire, A, B et C peuvent être fractionnaires ou négatifs.

2.4. L'ordre CALL

Cet ordre est utilisé pour l' appel de sous-programmes et possède la syntaxe :

```
CALL NAME(PARA 1,...PARA N)
```

instruction dans laquelle les paramètres peuvent être des valeurs numériques, représenter une valeur de chaîne ou correspondre à une référence. Par exemple, l'instruction :

```
CALL TT10ST(10)
```

peut établir la ligne de faisceau TT10 à 10 GeV. Les sous programmes seront normalement utilisés pour prendre des valeurs dans le programme principal ou les y réintroduire, par exemple :

```
CALL COACQ(PLANE, TIMING, TABLE)
```

Les deux premiers paramètres peuvent être des paramètres numériques définissant le plan et la synchronisation pour l'acquisition d'une orbite fermée, alors que TABLE peut être un paramètre de référence définissant un tableau dans lequel les résultats doivent figurer.

Les sous-programmes peuvent être écrits en NODAL (chapitre 8) et en langage d'assemblage (chapitres 6 et 7). Dans les ordinateurs de pupitre, la plupart des sous-programmes concernant les processus seront rédigés en NODAL, alors que dans les ordinateurs locaux la plupart d'entre eux seront écrits en langage d'assemblage.

L'emploi du mot-clé CALL est facultatif. Ainsi on peut écrire :

```
NAME(PARA1, ... PARAN)
TT10ST(10)
COACQ(PLANE, TIMING, TABLE)
```

2.5. Organisation des programmes en suites d'ordres

Jusqu'à maintenant nous n'avons discuté que des ordres immédiatement exécutés par NODAL. Etant donné que le système NODAL est un système interprétatif, ces ordres sont compris et exécutés immédiatement, sans compilation intermédiaire. Toutefois, si la ligne d'instructions comprend en préfixe un numéro de ligne, les ordres ne sont pas exécutés immédiatement, mais ils sont stockés en vue d'une exécution ultérieure, réalisée habituellement dans le cadre d'une suite d'ordres.

Les numéros des lignes doivent être compris dans l'intervalle 1.01 à 99.99. Les nombres 1.00, 2.00, etc, sont illégaux car ils sont utilisés pour identifier le groupe entier. Le nombre situé à la gauche du point est appelé numéro de groupe.

```
1.10 SET A=1
1.30 SET B=2
1.50 TYPE A+B
```

le programme mémorisé peut être exécuté en frappant RUN sur le clavie Lorsqu'il a été mis au point, il peut être stocké dans un fichier et exécuté en utilisant l'ordre RUN, ou à la suite d'une interruption, où à un instant déterminé, ou encore à des instants régulièrement espacés.

2.5.1 L'ordre RUN

Cet ordre lance l'exécution du programme. RUN par lui-même lance l'exécution d'un programme qui vient d'être frappé au clavier en commençant par la ligne ayant le plus bas numéro. Toutefois :

```
RUN [2.7]
```

lance l'exécution à la ligne 2.7 . On notera qu'ici : 2.7 = 2.70

RUN FILENAME

entraîne le chargement du programme stocké dans le fichier FILENAME et son exécution.

RUN [X] FILENAME

entraîne le chargement du programme stocké dans FILENAME, alors que l'exécution commence à la ligne dont le numéro correspond à la valeur de l'expression X.

2.5.2 L'ordre DO

L'ordre DO est utilisé pour transférer le contrôle à une ligne ou à un groupe de lignes déterminés, avec retour automatique à l'instruction suivant l'ordre DO. Par exemple le programme :

```
1.1 SET A=1; SET B=2
1.2 TYPE "STARTING = "
1.3 DO 3.2

2.1 TYPE "FINISHED = "

3.1 SET A=3; SET B=4
3.2 TYPE A+B
```

donne lorsque l'on a frappé RUN :

STARTING = 3 FINISHED = 7

La structure en groupes de NODAL est utile pour diviser un programme en sous-unités logiques ou en sous-programmes. On peut affecter un numéro de groupe différent à chacun de ces modules, qui seront alors exécutés à l'aide de l'ordre DO.
L'ordre DO est normalement inséré dans une séquence du programme, par exemple :

```
1.1 FOR INJPHS = 5,15; DO 2
1.2 END
```

```
2.1   TYPE INJPHS
2.2   TYPE BCT (3)
```

L'ordre DO peut également être donné directement depuis le clavier, de sorte que l'on peut tester individuellement des parties d'un programme. Cela est utile étant donné que de nombreux programmes comprennent une partie acquisition, une partie calcul, puis une partie commande.

Dans l'ordre DO, le champ correspondant au numéro de la ligne peut être toute expression valide.
Par exemple :

```
DO BUTTON + 1
```

peut entraîner l'exécution du groupe 4 si l'on frappe button 3.
Cette possibilité offre aussi une puissante ressource calculée du type "CASE".

2.5.3 L'ordre RETURN

L'ordre RETURN est utilisé pour sortir d'un sous-programme DO. Lorsqu'un ordre RETURN est rencontré au cours de l'exécution d'un sous-programme DO, le programme quitte le mode sous-programme et revient à l'instruction suivant l'ordre DO qui a lancé le mode sous-programme.

2.5.4. L'ordre GOTO

En NODAL, l'ordre GOTO transfère le contrôle à une ligne déterminée dans un programme en cours d'exécution. Après exécution de l'instruction correspondante, NODAL passe à la ligne de numéro immédiatement supérieur. Le GOTO entraîne donc un branchement du programme, c'est-à-dire un saut à une ligne précédente ou ultérieure; par exemple :

```
GOTO 1.3
```

Le numéro de la ligne peut être remplacé par toute expression valide en NODAL. Cela correspond à un GOTO calculé, par exemple :

```
GOTO 4-2.7
```

donne le même résultat que GOTO 1.3.

Il n'est pas possible de sortir d'un groupe DO en utilisant l'ordre GOTO. Toute tentative de ce genre sera traitée comme un ordre RETURN. Il y aura donc lieu de faire attention lorsque la cible d'un GOTOse situera à l'extérieur du groupe d'instructions actuel. Cela n'est permis qu'à la condition que les groupe fassent partie du programme principal et ne soient jamais appelés en tant que sous-programmes DO.

2.5.5 Les ordres END et QUIT

Un programme se termine normalement lorsqu'il a atteint l'extrémité de la ligne ayant le numéro le plus élevé. On peut provoquer un arrêt antérieur en utilisant l'ordre END. Dans le mode en ligne, END entraîne le retour du contrôle au terminal en ligne. Dans des programmes avec interruptions, END entraîne un retour du contrôle au moniteur, et le programme disparait. QUIT provoque toujours un retour au moniteur et NODAL est alors déconnecté.

2.5.6 L'ordre %

Lorsque l'on place le symbole % au début d'une suite d'instructions, le reste de cette ligne est ignoré de sorte que l'on peut insérer des commentaires dans le programme. Les lignes correspondantes seront sautées lors de l'exécution du programme, mais un ordre LIST permettra de les afficher.

2.6 Prise de décisions : les ordres IF et WHILE

La possibilité de prendre des décisions représente l'une des caractéristiques les plus importantes d'un programme traité sur ordinateur. En NODAL la prise de décision se réalise en utilisant les ordres IF et WHILE.

2.6.1 Le IF arithmétique

Cet ordre permet de contrôler les sauts de programme, d'après le signe d'une expression arithmétique. Par exemple :

```
IF (A-10) 3.2 ; DO 2
```

Si la valeur de l'expression entre parenthèse est négative, c'est-à-dire, ici, si A est inférieur à 10, le contrôle est transféré à la ligne 3.2. Dans le cas contraire, l'instruction suivante (ici : DO 2) est exécutée.

```
IF (X)3.2,3.3; DO 2
```

Dans cet exemple si X est négatif, le contrôle est transféré à 3,2 ; si X est nul, le contrôle est transféré à 3.3 ; enfin si X est positif, l'instruction suivante - ici, DO 2 - est exécutée.

```
IF (X)3.1,3.2,3.3
```

Dans cet exemple il y a toujours branchement vers 3.1, 3.2 ou 3.3 selon que X est négatif, nul ou positif.

Des ordres IF peuvent être associés afin de réaliser des conditions plus complexes, par exemple :

```
2.1 IF (A)3.1; IF (10-A)3.1,3.1; DO 4
```

Dans ce cas le groupe d'instructions 4 est exécuté si $0 \leq A < 10$; dans le cas contraire, le contrôle est transféré à la ligne 3.1.

L'expression entre parenthèses est considérée comme égale à zéro si sa valeur se situe dans l'intervalle $\pm$ 10 E - 6.

2.6.2 Le IF logique

Cet ordre permet d'exécuter le reste d'une ligne d'instructions à la condition qu'une expression logique soit valide, par exemple :

```
IF A<132; TYPE "LESS THAN"; DO 4
```

Si la condition est satisfaite, c'est-à-dire, ici, si A est inférieur à 132, le reste de la ligne est exécuté. Dans le cas contraire ce même reste est ignoré. L'expression logique est formée de deux expressions arithmétiques séparées par l'un des opérateurs logiques ci-après :

> plus grand que
< plus petit que
= égal à
>= supérieur ou égal à
<= inférieur ou égal à
<> non égal à

Afin d'éviter la confusion avec l'ordre IF arithmétique décrit précédemment, la première expression arithmétique ne doit pas être mise entre parenthèses. Les deux expressions sont considérées comme égales si leur différence est inférieure à $\pm$ 5 E - 8, par rapport à la première.

2.6.3. L'ordre WHILE

Cet ordre est analogue au IF logique ci-dessus, à ceci près que le reste de la ligne est exécuté en boucle jusqu'à ce que la condition ne soit plus satisfaite. Par exemple :

```
WHILE CAMAC(0,1,0,0)=0; SET CAMAC(0,2,0,16)=1
```

provoque l'écriture répétée d'un 1 dans le module 2, jusqu'à ce que le module 1 prenne la valeur 1.

2.7. Sous-programmes pour les variables du système

NODAL est relié à l'équipement de l'accélérateur par l'intermédiaire de sous-programmes écrits en code machine.

Ces sous-programmes peuvent être de deux types : variables du système et sous-programmes CALL.

Les variables du système représentent le coeur des caractéristiques en ligne de NODAL. Elles permettent de manipuler les paramètres de la machine avec la même puissance et la même souplesse que les variables habituelles du programme. Toutefois, la commande et l'acquisition s'effectuent par l'intermédiaire d'un sous-programme écrit par l'utilisateur, au lieu de dépendre simplement de la mémoire de l'ordinateur. Afin d'éviter de remplir cette mémoire de nombreux sous-programmes analogues, on utilise des paramètres. Par exemple une variable du système peut effectuer tout le travail pour toutes les pompes ioniques pulvérisation, soit :

```
VPS(N,P)
```

où N est le numéro de la pompe et P, la propriété de la pompe considérée. Ainsi :

```
TYPE VPS(40,#CUR)
```

peut fournir l'intensité de courant dans la pompe 40 ;

```
SET VPS(40,#SWI) = 1
```

peut fermer l'interrupteur de la pompe 40. Les variables du système peuvent agir de deux manières : en premier lieu elles peuvent accéder directement à l'équipement; en second lieu, elles peuvent simplement accéder à des tables utilisées ou remplies par une tâche autonome. Il est même possible de combiner ces deux modes d'action; par exemple :

```
TYPE VPS(40,#CUR)
```

peut accéder au véritable courant circulant dans la pompe, alors que :

```
TYPE VPS(40,#CUS)
```

peut accéder à la dernière des valeurs de ce paramètre, obtenues par échantillonnage.

Les sous-programmes CALL assurent un lien plus général 'entre le programme en NODAL et l'équipement. Elles sont appelées à l'aide d'une instruction de la forme :

```
CALL NAME (PARAMETER 1,....)
```

Les paramètres ou arguments du CALL ci-dessus ou des sous-programmes avec variables du système pouvant être de trois types : valeur numérique, valeur de chaîne ou référence. Les paramètres à valeur numérique sont utilisés pour transférer un nombre dans un sous-programme. Dans la séquence d'appel, les paramètres sont utilisés pour transférer une chaîne de caractères au sous-programme; ils prennent la forme d'une expression de chaîne. Les paramètres de référence sont utilisés pour transmettre au sous-programme l'adresse d'éléments de données (par exemple, des tableaux) de NODAL. Dans la séquence d'appel on utilise un nom de NODAL. Par exemple, avec l'instruction :

```
CALL NAME(3, "YES", ABC)
```

Le sous-programme NAME acquiert le nombre 3, la chaîne de caractères YES et l'adresse du tableau ABC.

2.8. Données complémentaires sur les entrées et sorties

2.8.1 L'ordre TYPE

Cet ordre a été introduit au paragraphe 2.1, et il a été illustré par la suite à l'aide de plusieurs exemples.

La forme générale de l'instruction est :

TYPE typelist

ou "typelist" représente une série d'éléments primaires. Des éléments primaires sont des expressions dont la valeur est imprimée, des chaînes de caractères (par exemple : "THIS IS") dont le texte compris entre guillements est imprimé, ou des caractères de contrôle tels que ! , pour une nouvelle ligne. Par exemple :

```
TYPE "VALUES ARE" A B ! "END"
```

peut fournir le libellé :

```
VALUES ARE   2.5362      4.1234
END
```

Les caractères de contrôle provoquent l'exécution de tâches particulières par NODAL. Ces caractères sont les suivantes (X représente une expression légale quelconque) :

! Inscrire sur une nouvelle ligne (retour de chariot et avance d'une ligne)

%X Changer de format

%, Utiliser le format E

\X Afficher l'équivalent de X en ASCII

&X Afficher X espaces

]X Afficher X en octal

?X Afficher X en binaire

Le format normal en virgule fixe pour les valeurs numériques comprend 6 chiffres avant le point décimal et quatre chiffres après, c'est-à-dire qu'il occupe un champ total de 11 positions. Ce format peut être modifié en utilisant le caractère de contrôle %. Si on insère la séquence de contrôle % 8.03 dans la liste de l'instruction faisant suite à l'ordre TYPE, le format est modifié et correspond alors à 3 chiffres après le point décimal dans un champ ayant une longueur de 8 caractères. % commande l'écriture dans le format E; par exemple : TYPE %, 1 imprime : 1.000 000 00 E0. Il faut noter que cet emploi du symbole % n'intervient que dans une liste suivant l'ordre TYPE, de sorte qu'il n'y a pas de confusion avec l'ordre % (utilisé pour l'impression de commentaires). Si le nombre est trop grand pour tenir dans le format spécifié, le format E est alors utilisé automatiquement Une écriture sous forme de nombre entier est obtenue lorsque le symbole % est suivi d'un entier. Par exemple :

```
TYPE %5 A+B
```

imprimera la valeur de A + B sous la forme d'un entier justifié à droite, dans un champ de 5 positions.

Des virgules peuvent être utilisées pour séparer des éléments dans la liste d'un ordre TYPE, par exemple :

```
TYPE  A  B
```

est équivalent à

```
TYPE A,B
```

mais

```
TYPE A -B
```

n'est pas équivalent à

```
TYPE A, -B
```

2.8.2 L'ordre ASK

L'ordre ASK est normalement utilisé dans des instructions indirectes afin de permettre à l'utilisateur d'introduire des données en des points déterminés, au cours de l'exécution de son programme. L'ordre ASK est écrit sous la forme :

```
11.99  ASK  X   Y Z
```

Lorsque NODAL rencontre la ligne 11.99, il imprime deux points (:). L'utilisateur peut alors frapper au clavier une expression dont la valeur sera attribuée à la variable X. Si l'on frappe seulement un astérisque (*), la valeur de X n'est pas modifiée. Pour que l'évaluation se réalise, il faut effectuer un retour de chariot. S'il est demandé de fournir des valeurs à plus d'une variable (par exemple X, Y, Z, ci-dessus), les valeurs correspondantes peuvent être frappées sur une même ligne, en les séparant par des virgules.

Un texte peut être inséré dans une instruction ASK, par exemple :

```
11.99 ASK  "VALUE OF X"   X  "VALUE OF Y"  Y  " VALUE OF Z" Z
```

Lorsque NODAL exécute l'instruction de la ligne 11.99, il commence par afficher :

```
VALUE OF X :
```

L'utilisateur peut par exemple répondre ainsi : 2 < retour de chariot > , puis 3 < retour de chariot > et enfin 4 < retour de chariot > ce qui peut donner le libellé :

```
VALUE OF X : 2
VALUE OF Y : 3
VALUE OF Z : 4
```

Les variables X, Y et Z ont maintenant respectivement les valeurs 2, 3 et 4. Si l'utilisateur sait que les valeurs de Y et Z seront demandées immédiatement après celle de X, il peut frapper au clavier les trois valeurs en une seule fois, en les séparant par des virgules, soit :

```
VALUE OF X : 2, 3, 4
```

ce qui supprime l'impression inutile des deux lignes suivantes.

2.9 Erreurs, ordres LIST et ERASE, ressource pour l'affichage des étapes intermédiaires du traitement

L'une des caractéristiques de NODAL consiste en la détection des erreurs d'un programme au moment de son exécution, avec des possibilités très développées de détection des erreurs et d'édition ultérieure du programme.

2.9.1 Détection des erreurs

Lorsqu'une erreur se présente au cours de l'exécution d'une instruction, un message d'erreur est imprimé, par exemple :

NON EXISTENT NAME AT LINE 2.2

NODAL s'arrête et passe dans le mode commande, de sorte que la ligne comportant une erreur peut être corrigée. A titre d'exemple, on peut considérer la séquence d'opérations ci-après :

```
>1.1  SET A=2; TYPE "A" A !
>1.2  SET B=4; TYPE "B" B !
>1.3  TYPE AB+B
```

RUN

A 2

B 4

NON EXISTENT NAME AT LINE 1.30

(Note : le symbole > est imprimé par NODAL pour indiquer qu'il attend la frappe d'une ligne ou d'un ordre immédiat).

Dans le cas de longues lignes, l'utilisateur peut obtenir une meilleure indication sur la localisation de l'erreur en frappant au clavier CTRL/B (on continue d'appuyer sur la touche CTRL pendant que l'on frappe B). NODAL imprime alors la ligne incriminée, avec une flèche indiquant l'endroit de l'erreur, c'est-à-dire que dans l'exemple précédent, à la suite de la frappe de CTRL/B, on obtiendrait

```
1.30  TYPE  AB + B
            ↑
```

Maintenant NODAL est automatiquement passé en mode EDIT et il est prêt à modifier la ligne 1.3.

2.9.2 L'ordre LIST

L'ordre LIST par lui-même, entraîne l'affichage sur le terminal de la totalité du programme par NODAL. Toutefois, LIST 1.1 ou LIST 2 ne provoquera que l'impression de la ligne 1.1 ou des lignes du groupe 2. L'instruction LIST 1.1 2 3 entraîne l'affichage de la ligne 1.1 et des lignes des groupes deux et trois.

2.9.3 L'ordre ERASE

Des lignes, des groupes, et des variables peuvent être supprimés dans la liste introduite par l'utilisateur, en employant l'ordre ERASE ; par exemple :

ERASE 1.1 2 A B

provoquera la suppression de la ligne 1.1, du groupe 2 et des éléments des données A et B. L'ordre :

ERASE ALL

vide la mémoire de l'utilisateur; il doit être utilisé avant de frapper au clavier un second programme à la suite d'un premier, de manière à éviter un mélange entre ces deux programmes.

2.9.4 La ressource d'affichage d'étapes intermédiaires

Lorsque l'on frappe l'ordre ?ON, chaque ligne d'un programme est affichée avant exécution, de sorte que l'utilisateur peut observer la progression du traitement de son programme. L'instruction ?OFF déconnecte cette ressource.

2.10 L'ordre EDIT

La manière la plus simple d'éditer une ligne est de la frapper à nouveau au clavier. Dans ce cas NODAL insère la nouvelle ligne à la place de l'ancienne, en lui affectant le même numéro, c'est-à-dire qu'il effectue un ERASE automatique.

Toutefois, l'ordre EDIT offre de puissantes possibilités pour l'édition des lignes, qu'il s'agisse des lignes anciennes déjà mémorisées dans l'ordinateur ou des lignes nouvelles, à la suite de leur frappe. Cette ressource est identique au sous-programme d'édition de lignes qui est inséré dans QED, le programme d'édition des ordinateurs NORD-10.

L'ordre EDIT possède le format suivant :

EDIT NN.NN , L

où NN.NN est le numéro de la ligne à éditer. Le symbole L indique qu'un affichage de la ligne a été requis avant le passage en mode édit, mais il est facultatif. L'ordre EDIT opère une transformation caractère par caractère de l'ancienne ligne NN.NN pour donner la nouvelle ligne qui doit être créée. Des caractères spéciaux sont utilisés pour contrôler cette transformation. Il s'agit de caractères de contrôle, tels que par exemple CTRL/C, représentant le contrôle C. Ces caractères sont obtenus en continuant d'appuyer sur la touche CTRL pendant que l'on frappe le caractère requis.

Les commandes de transformation ponctuelle des plus évidentes sont les ordres de copie; par exemple CTRL/C copie un caractère de

l'ancienne ligne dans la nouvelle ligne. CTRL/O Character où character désigne un caractère isolé quelconque, copie tous les caractères de la ligne jusqu'à Character exclus. L'ordre EDIT s'exécute normalement dans le mode remplacement de sorte que tout caractère autre que CTRL se trouve inséré dans l'ancienne ligne. Par exemple, pour éditer la ligne

```
1.1 LET A=1
```

qui devrait s'écrire :

```
1.1 SET A=1
```

on peut frapper EDIT 1.1, puis CTRL/O L qui copie la ligne jusqu'à L exclus; on frappe ensuite S qui remplace L puis CTRL/D qui copie le reste de l'ancienne ligne et achève ainsi l'exécution de l'ordre EDIT.

Il est également possible d'insérer des caractères en un point quelconque d'une ligne, en frappant CTRL/E. NODAL imprime le symbole < , puis insère tous les caractères qui viennent d'être frappés, jusqu'à ce que l'on frappe un autre CTRL/E auquel il est répondu par affichage du symbole > . Supposons par exemple que nous frappions EDIT 1.1 , L qui fournit :

```
1.1 SET A=1
```

alors que nous voudrions obtenir :

```
1.1 SET A123=1
```

Pour ce faire, nous pouvons frapper CTRL/Z A, qui copie l'ancienne ligne jusqu'à A inclus, puis CTRL/E qui contrôle l'insertion, puis 123, puis à nouveau CTRL/E qui termine l'insertion, et enfin CTRL/D qui copie le reste de l'ancienne ligne, afin d'achever l'édition. Il est également possible de sauter au-delà de caractères dans l'ancienne ligne qui ne nous intéressent pas. Par exemple, CTRL/S saute un caractère de l'ancienne ligne, alors que CTRL/X C saute toute une portion de ligne, jusqu'au caractère C inclus. Lorsque l'on édite de longues lignes il peut se faire que l'on oublie ce qui a été effectué auparavant. Dans ces conditions, CTRL/Y copie le reste de l'ancienne ligne et relance l'édition. Si l'on veut observer le contenu de la ligne, on peut frapper CTRL/H qui copie et affiche l'ensemble des caractères jusqu'à la fin de la ligne. En frappant à nouveau CTRL/Y on redémarre l'édition de cette ligne totalement imprimée en clair.

Lorsque l'on frappe une nouvelle ligne ou que l'on édite une ancienne ligne, on peut faire des erreurs. En frappant CTRL/A, on supprime le dernier caractère de la nouvelle ligne et on obtient en réponse le symbole ↑ . CTRL/W supprime le dernier mot; il lui est répondu par \. CTRL/Q supprime la nouvelle ligne en entier. Il y est répondu par une flèche dirigée vers l'arrière, puis il se produit un retour de chariot avec avance d'une ligne, de sorte que l'on peut recommencer. Toutefois ce dernier contrôle n'a pas besoin d'être très souvent utilisé, car le fait de frapper un retour de chariot termine l'édition et insère la nouvelle ligne même si elle comporte des erreurs. Cette ligne devient automatiquement l'ancienne ligne ; elle peut donc être éditée et corrigée, à la condition que son numéro ne soit pas modifié.

D'autres caractères de contrôle sont disponibles pour d'autres tâches d'édition; une liste complète en est donnée à l'ANNEXE II.

2.11 Tableaux et fonctions mathématiques

2.11.1 Tableaux

On crée des tableaux à l'aide de l'ordre DIMENSION, par exemple :

```
DIM   A(36)
```

créera un tableau de nombres écrits en virgule flottante, et contenant trente-six mots auxquels on aura accès à l'aide des expression A (1)... A(36). Lors de la définition de la dimension, les éléments des tableaux sont initialisés à zéro. On peut également créer des tableaux de nombres entiers; par exemple :

```
DIM-INT  A(X)
```

créerait un tableau de nombres entiers contenant X mots. On peut créer des tableaux à deux dimensions des deux types, à l'aide, par exemple, de :

```
DIM   B(10,2)
DI-I  J(4,5)
```

Des tableaux de chaînes de caractères sont créés en utilisant l'ordre DIMENSION-STRING, soit :

```
DIM-S  STR
```

Comme dans le FORTRAN IV du NORD, les tableaux sont mémorisés dans l'ordre ascendant des positions mémoire. Les tableaux à deux dimensions sont mémorisés colonne par colonne, c'est-à-dire que c'est le premier indice qui varie en premier. Par exemple le tableau A(3,2) est mémorisé sous la forme :

A(1.1)
A(2.1)
A(3.1)
A(1,2)
A(2,2)
A(3,2)

Dans un programme, on peut accéder aux éléments d'un tableau à deux dimensions en le considérant comme un tableau à une dimension. Ainsi, dans l'exemple précédent A(4) est équivalant à A(1,2). Une vérification est effectuée sur les limites des tableaux afin d'éviter qu'une erreur de programmation entraîne une écriture à l'extérieur d'un tableau.

2.11.2 Fonctions mathématiques

Elles ont la signification habituelle et il s'agit en fait simplement de variables du système du type mathématique plutôt que du type paramètre de l'accélérateur. Par exemple :

```
SET Y = SIN(X)
```

a une signification évidente. Les fonctions disponibles sont les suivantes :

Nom de la fonction	Valeur qu'elle fournit
SIN(X)	sinus de X, X étant exprimé en radians
COS(X)	cosinus de X, X étant exprimé en radians
AT2 (X,Y)	arc tangente de Y/X dans l'intervalle 0-2π
SQR(X)	racine carrée de X
EXP(X)	exponentielle de X
LOG(X)	logarithme néperien de X ($LOG_e(X)$)
ABS(X)	valeur absolue de X
INT(X)	partie entière de X
FPT(X)	partie fractionnaire de X (même signe que X de sorte que : INT(X) + FPT(X) = X)
SGN(X)	$\pm$ 1, selon le signe de X
MOD(X,Y)	X modulo Y.

3. EMPLOI DE FICHIERS

L'une des caractéristiques de NODAL tient au fait que les programmes et les données sont essentiellement translatables et indépendants de la machine. Cela permet d'utiliser intégralement la souplesse d'un système de fichiers à usage général en vue de la mémorisation des programmes, des recouvrements de programmes, etc. Dans ce qui suit "FILENAME" représente le nom d'un fichier quelconque faisant partie du système de fichiers utilisé. Il peut s'agir d'un nombre, par exemple 100 dans un système de base SINTRAN, d'un nom NODAL, comme dans le cas du système temporaire décrit dans le document LAB-CO/INT/Comp.Note/74-2, ou d'un nom de fichier complet pour un système exploité en temps partagé. Pour d'autres types d'ordinateurs, il s'agira d'un spécificateur approprié pour un fichier, par exemple : DT1:FRED pour le DOS d'un PDP-11.

3.1 Conservation des programmes et des données

Des programmes et des données peuvent être stockés en vue d'une consultation ultérieure, en utilisant l'ordre SAVE, par exemple :

```
SAVE FILENAME
```

stockera le programme en cours dans le fichier appelé FILENAME. Des données peuvent également être conservées, par exemple :

```
SAVE FILENAME ABC
```

stockera le tableau ABC dans le fichier FILENAME. Des programmes et des données peuvent être simultanément mémorisés, par exemple :

SAVE FILENAME 2 ABC

stockera le groupe 2 du programme en cours ainsi que le tableau ABC dans le fichier FILENAME.

Les éléments qui ont été stockés peuvent être extraits en utilisant l'ordre LOAD, par exemple :

LOAD FILENAME

Le contenu du fichier FILENAME est chargé et inséré dans le programme en cours. Si FILENAME contient des lignes de programme ou des éléments de données ayant le même nom que ceux qui existent déjà dans ce programme, ces derniers sont supprimés et les nouveaux éléments sont insérés à leur place.

L'ordre OLD est équivalent à un ordre ERASE ALL suivi d'un ordre LOAD, par exemple :

OLD FILENAME

supprime le contenu actuel de la zone affectée à l'utilisateur, puis il y charge le contenu du fichier FILENAME.

L'ordre RUN décrit au paragraphe 2.4, ci-dessus, correspond en fait à un ordre OLD suivi d'un ordre RUN.

3.2. Variables globales de tâches

Dans des configurations rencontrées dans de petits ordinateurs le tampon de NODAL peut être trop petit pour contenir le programme complet. L'une des manières de surmonter cette difficulté consiste à realiser un chaînage du programme. Celui-ci est divisé en plusieurs parties successives dont chacune est mémorisée dans un fichier séparé, par exemple FILE1 , FILE2, FILE3. Ensuite, pour traiter le programme on frappe "RUN FILE1" qui entraîne l'exécution de la première partie du programme. Le travail se poursuit avec l'ordre "RUN FILE2" qui provoque le chargement et l'exécution de la seconde partie, et ainsi de suite.

Toutefois il est habituellement nécessaire d'échanger des données entre les différentes parties d'un programme. S'il y a lieu d'échanger un grand nombre de données, par exemple celles d'un tableau ABC, il faut utiliser un autre fichier. Dans le fichier FILE1, l'avant-dernier ordre peut être alors "SAVE FILE4 ABC" qui placera le tableau ABC dans le fichier FILE4. Dans ces conditions, la première instruction figurant dans FILE2 peut être "LOAD FILE4" de sorte que le tableau ABC sera chargé exactement dans son état actuel.

Il arrive également souvent que seules quelques valeurs ont à être échangées entre des programmes. A cette fin il a été prévu 16 variables globales de tâches appelées ARG(1)...ARG(16). Elles conservent leurs valeurs tant que la tâche NODAL considérée est active, c'est-à-dire tant qu'elle n'est pas affectée par un ordre ERASE ALL, OLD, SAVE, LOAD, etc. Les valeurs de ces variables sont établies ou lues à l'aide des instructions :

```
SET ARG(5) = X
SET X = ARG(5)
```

Cela permet de créer facilement des suites de programmes chaînés du type "RUN FILE", en stockant les principaux paramètres échangés à l'aide des variables globales ARG(1)...ARG(16).

3.3. Sous-programmes sur fichiers

NODAL assure plusieurs niveaux de structure pour les sous-programmes. Naturellement on peut utiliser des sous-programmes repérés par des noms différents, comme cela sera discuté ultérieurement. Mais NODAL permet également d'utiliser comme sous-programmes des lignes et/ou des groupes de lignes; par exemple :

```
1.1 DO 2.1; DO 3; DO X; QUIT
```

est une ligne contenant trois appels de sous-programmes, correspondant à la ligne 2.1, au groupe 3 et à la ligne ou au groupe dont le numéro est donné par la valeur de X.

Il est souvent commode d'être en mesure d'écrire et de stocker séparément des sous-programmes. Cela peut être réalisé en utilisant les ordres SAVE et LOAD discutés dans le paragraphe précédent, étant donné qu'une ligne ou un groupe de lignes peuvent être stockés dans un fichier puis chargés et exécutés à l'aide d'un ordre DO. Toutefois il s'agit plus effectivement d'une reconfiguration dynamique du programme principal plutôt que de l'appel d'un sous-programme indépendant.

Pour une telle application, une suite d'instructions typique peut s'écrire :

```
1.1  % MAIN PROGRAM TO LOAD AND EXECUTE GROUPS FROM FILES
1.2  LOAD FILE1 ; DO 10 ; SET A=3 ; DO 11
1.3  ERASE 10 11 12 ; LOAD FILE2 ; DO 30 ; END
```

Cette suite charge des groupes stockés dans les fichiers, puis les exécute à l'aide des ordres DO. L'ordre LOAD est normalement conçu pour le chargement d'éléments de données, mais il peut également être utilisé comme ci-dessus pour le chargement d'éléments d'un programme dans le tampon en les combinant avec les éléments qui y sont déjà présents. Toutefois, les lignes chargées au cours de l'exécution d'un programme doivent avoir un numéro plus élevé que toute ligne actuellement active.

3.4. La ressource OVERLAY

Dans le cas d'un sous-programme stocké sur fichier, l'ordre OVERLAY permet de le charger et de l'exécuter au milieu d'un programme principal, en ne perturbant aucunement celui-ci. Le sous-programme peut utiliser les mêmes numéros de lignes ou les mêmes variables que le programme principal sans qu'il y ait interférence. Le recouvrement du sous-programme partage temporairement le tampon avec le programme principal, de sorte que la somme de leurs longueurs doit être adaptée à la capacité du tampon. Toutefois le sous-programme disparaît après utilisation et le programme principal peut alors appeler d'autres recouvrements. Des recouvrements peuvent être imbriqués jusqu'à toute profondeur permise par la capacité du tampon affecté au traitement de la tâche. Ainsi, dans un certain programme, la ligne numérotée par exemple 5.7 :

```
5.7  TYPE MBBH(1) ; OVERLAY FILENAME ; TYPE MBBH(1)
```

entraînera l'affichage de la valeur de MBBH(1), puis un certain programme stocké dans le fichier FILENAME sera traité, puis la valeur de MBBH(1) sera à nouveau imprimée.

Les communications entre le programme principal et les modules avec recouvrement peuvent être réalisées en utilisant les variables globales de tâches ARG(1)...ARG(16), comme cela a été décrit au paragraphe 3.2. Il s'agit là d'une possibilité tellement utile qu'elle a été insérée pour exécution automatique dans l'ordre OVERLAY; par exemple :

OVERLAY FILENAME 2,3,4

Cet ordre exécute comme précédemment le module à recouvrement FILENAME, mais commence par donner aux variables globales les valeurs ci-après : ARG(1) = 2, ARG(2) = 3 et ARG(3) = 4. Ces valeurs peuvent être prélevées et utilisées comme on le souhaite dans le module FILENAME. Seules les huit premières variables ARG peuvent être automatiquement établies de cette manière. Toutes expressions légales, séparées par des virgules, peuvent être utilisées pour spécifier les valeurs des ARG.

4. RESSOURCES POUR L'EXPLOITATION EN TEMPS REEL

Etant donné que NODAL est un langage conçu pour la commande en temps réel de l'accélérateur, des ressources y sont incorporées afin de synchroniser les programmes avec le séquencement de l'accélérateur ou avec une horloge. Des programmes peuvent également être ordonnancés en vue de les traiter à des instants déterminés ou chaînés à des interruptions.

4.1 L'ordre WAIT

Les programmes sont synchronisés à l'aide de l'ordre WAIT qui se présente sous trois formes :

```
WAIT-TIME TIME
WAIT-CYCLE EVENT,DELAY
WAIT (computer number)
```

La première instruction impose au programme d'attendre un nombre donné de secondes avant de passer à l'instruction suivante. Les ordres :

```
SET MBBH(1)=456 ; WAIT-TIME 0.5 ; TYPE MBBH(1)
```

entraînent l'établissement d'une intensité de courant par l'ordinateur, l'attente d'une demi-seconde et l'affichage de l'intensité.

La seconde forme impose à l'ordinateur d'attendre un certain temps au cours d'un cycle de l'accélérateur avant de continuer le traitement. Ce temps est défini à l'aide d'un événement et du temps compté à partir de cet événement. Cette dernière indication peut être omise si elle n'est pas requise.

La troisième forme indique à l'ordinateur d'attendre des données qui doivent être fournies par un autre ordinateur. Une discussion complémentaire de cette instruction figure à la section 5.

REMARQUE IMPORTANTE

L'emploi des deux premières formes de l'ordre WAIT peut entraîner un verrouillage pendant un long intervalle de temps de ressources précieuses, chargées dans l'ordinateur, bloquées ainsi par le programme en cours. Ces ressources ne doivent pas être utilisées dans des programmes en temps réel ayant une priorité élevée ou dans une suite d'ordres transmis pour exécution à un ordinateur éloigné, à l'aide de l'ordre EXECUTE (voir section 5).

4.2 Ordonnancement des programmes

Les programmes NODAL peuvent être traités sur trois "niveaux NODAL", soit : le niveau "dialogique", le niveau "temps réel" et le niveau "ordre immédiat". On ne doit pas les confondre avec les niveaux du système d'exploitation; la relation entre ces deux groupes de niveaux dépend du système considéré.

Le niveau dialogique est celui qui a été discuté jusqu'à maintenant. Les programmes exécutés à ce niveau sont traités en association avec un télétype en ligne et sont en liaison directe avec un opérateur. Des programmes contenus dans des fichiers peuvent être lancés en utilisant l'ordre RUN. Ils peuvent obtenir une réponse de l'opérateur par l'intermédiaire de l'ordre ASK.

Le niveau temps réel est conçu pour un autre type de programme. Il s'agit de programmes qui ont été rédigés, testés et stockés au niveau dialogique mais qui doivent être traités de manière autonome. Le lancement de leur exécution est commandé par une chronologie ou par une interruption et non pas par un opérateur. Leur seul rapport avec l'opérateur se réduit à l'impression d'un résultat ou plus probablement d'un message d'alerte. Des programmes en temps réel peuvent être ordonnancés pour vérifier certaines intensités de courant ou pressions, par exemple toutes les 15 minutes, [illegible], ou pour saisir certaines données au cours de chaque cycle et les stocker dans un fichier.

Le niveau ordre immédiat est discuté ultérieurement.

Des programmes stockés sur fichiers peuvent être ordonnancés en vue d'une exploitation en temps réel à l'aide des instructions suivantes :

```
SCHEDL(FILE,ODEV,TIME,INTERVAL)
RTRUN(FILE,ODEV,EVENT,DELAY)
REPEAT(FILE,ODEV,EVENT,DELAY)
HOOK(FILE,ODEV,LAM NO.)
```

L'ordre SCHEDL lance l'exécution au temps TIME d'un programme stocké sur le fichier FILE, avec répétition à chaque INTERVAL (il n'y a pas répétition si INTERVAL $\leq$ 0). ODEV spécifie le dispositif au niveau duquel se réalise la sortie. Les paramètres TIME et INTERVAL sont définis en secondes.

L'ordre RTRUN commande l'exécution du programme à un instant donné au cours du cycle de l'accélérateur; cet instant est défini à l'aide d'EVENT et de DELAY.

L'ordre REPEAT commande l'exécution du programme à l'instant spécifié, au cours de chaque cycle.

L'ordre HOOK entraîne l'exécution du programme à chaque fois qu'une interruption définie se produit.

L'ordre

DISCON(FILE)

déconnecte le programme stocké sur le fichier FILE, qui avait été activé par l'une quelconque des instructions SCHEDL, REPEAT ou HOOK. Un programme peut se déconnecter (DISCON) lui-même.

Le temps est mesuré en secondes, probablement depuis le 1er janvier à 0h00. Une fonction NODAL appelée "TIME" est disponible pour la lecture de ce temps. Ainsi pour commander l'exécution du programme stocké sur le fichier "MY:FIL", avec un seul traitement au bout d'un intervalle de 10 secondes, on peut utiliser l'instruction :

```
SCHEDL(MY:FIL, 1, TIME+10, 0)
```

D'autres fonctions seront fournies dans l'ordinateur de service pour la conversion des données calendaires en secondes et vice-versa. La fonction

LISP

commande à NODAL d'imprimer l'état des programmes chargés dans l'ordinateur, c'est-à-dire d'indiquer quels programmes sont respectivement sous la dépendance des ordres HOOK, SCHEDL, etc.

5. INTERACTION ENTRE ORDINATEURS

Pour le système de commande par ordinateur du SPS, il s'agit là de l'une des plus importantes caractéristiques de NODAL. Elle est exploitée à l'aide des quatre ordres IMEX, EXECUTE, REMIT, et WAIT.

5.1 L'ordre IMEX

IMEX (abrégé pour : exécution immédiate) provoque l'exécution d'un ordre dans un ordinateur éloigné avec retour de tout résultat à l'ordinateur donnant l'ordre, en vue d'un affichage. Par exemple :

IMEX (5) TYPE BCT(3)

entraîne la transmission de l'ordre TYPE BCT(3) à l'ordinateur 5, pour exécution immédiate. Le résultat, par exemple : 59.6, sera renvoyé à l'ordinateur initial et affiché. En fait le reste de la ligne est également transféré de sorte que l'on peut transmettre plusieurs ordres. Une suite typique peut se présenter ainsi :

```
>IM (5) TYPE BCT(3,#TIM) BCT(3)
  2563   59.6
>
```

5.2 L'ordre EXECUTE

C'est l'ordre principal pour une interaction programmée. Il est très analogue à l'ordre SAVE décrit au chapitre 3 à ceci près qu'au lieu de conserver des éléments NODAL dans un fichier, il les transmet à un autre ordinateur pour exécution. Par exemple :

EXECUTE (5) 2 ABC

transmettra le groupe 2 et le tableau ABC à l'ordinateur 5 en vue

de l'exécution sous forme de tâche en temps réel. Cet ordre est exploitable du fait que les éléments NODAL sont translatables et indépendants de la machine. De même les lignes NODAL ne sont pas traitées avant le lancement de l'exécution de sorte qu'elles peuvent être introduites et stockées dans des ordinateurs pour lesquels elles sont illégales, à la condition qu'elles soient transmises pour exécution à un autre ordinateur qui les considère comme légales.

5.3 L'ordre REMIT

Il arrive souvent que des programmes soient transmis à un autre ordinateur en vue d'obtenir une information en retour. Pour ce faire, on utilise l'ordre REMIT; par exemple :

```
REMIT A B
```

transmet les tableaux appelés A et B, en supposant que l'information nécessaire y a été insérée. REMIT signifie renvoyer, et cet ordre ne peut avoir de signification que dans le cadre d'une tâche transmise d'un ordinateur à un autre, en utilisant l'ordre EXECUTE. La destination de l'ordre REMIT est implicitement le programme ou sous-programme qui traite l'ordre EXECUTE. L'ordre REMIT est la dernière instruction exécutée dans une tâche EXECUTE, c'est-à-dire qu'il contient un END implicite.

5.4. L'ordre WAIT

L'ordre WAIT sous la forme :

```
WAIT (5)
```

entraîne la lecture dans le tampon affecté à la tâche des données retransmises par un ordre REMIT lié à une tâche EXECUTE. Le programme attend jusqu'à ce que ces données soient chargées dans le tampon.

S'il y a une erreur dans le programme EXECUTE, elle est signalée au moment de l'exécution de l'ordre WAIT; sinon l'erreur est indiquée au moment où le programme revient au mode commande.

5.5. Un exemple

La séquence d'instructions ci-dessous représente un programme qui peut être exécuté sur un ordinateur de pupitre afin d'obtenir une exploration de la position du faisceau éjecté, à l'entrée d'un septum. Elle illustre l'emploi des ordres EXECUTE, REMIT et WAIT :

```
1.1  ASK "INITIAL POSITION=" IP "FINAL POSITION=" FP
1.2  SET P=IP; EXECUTE (8) 3.2 P
1.3  FOR P=IP+1,FP; DO 2
1.4  END

2.1  WAIT-CYCLE 4;EXECUTE (8) 3 P; WAIT (8)
2.2  TYPE "POSITION=" P-1 "CHARGE=" A

3.1  SET A=MINSCN(2)
3.2  SET MINSCN(2,#PSN)=P
3.3  REMIT A
```

6. REDACTION DE MODULES EN CODE MACHINE POUR NODAL

Dans le passé NODAL a été décrit comme "un système d'appel de sous-programmes". Il est certain que la puissance et la vitesse de tout système NODAL dépendent dans une importante mesure de la bibliothèque de sous-programmes disponibles, écrits en code machine. De tels sous-programmes peuvent être appelés par NODAL selon l'une des trois modalités ci-après à l'aide de l'instruction d'appel :

```
CALL FFT(A,-1)
```

ou à l'aide de fonctions de lecture, par exemple :

```
SET Z=CAMAC(C,N,A,F)
```

ou sous la forme de fonctions d'écriture, par exemple :

```
SET MBBH(1,5) = 217
```

L'incorporation de ces deux derniers modes fait de NODAL un véritable langage de commande, étant donné que l'on a accès aux éléments de l'accélérateur par l'intermédiaire de sous-programmes en code machine, de manière exactement identique à l'accès aux variables du programme ou aux tableaux. Parmi les types d'éléments NODAL (voir Annexe 6), onze d'entre eux sont en relation avec des modules rédigés en code machine. Il s'agit des éléments ci-après

Type 6 : Fonctions mathématiques

par exemple :
```
SET Y = SIN(X)
SET Z = MOD(X,Y)
```

Ce sont des fonctions de lecture, avec un ou deux arguments à valeur réelle.

Type 7 : Variables du système

par exemple : SET MBBH(1,5) = 217
SET Z = MBBH(1,7)

Ce sont des fonctions de lecture-écriture avec zéro, un ou deux arguments à valeur entière.

Type 8 : Fonctions de lecture-écriture

par exemple : SET CAMAC(C,N,A,F) = 512
SET Z = CAMAC(C,N,A,F)

Ce sont des fonctions de lecture-écriture pouvant avoir jusqu'à 8 arguments dans une gamme étendue de types différents qui seront décrits.

Type 9 : Fonctions d'écriture seule

La fonction d'écriture représente un sous-ensemble du type 8

par exemple : SET VLT = 50

Type 10 : Fonctions de lecture seule

Il s'agit encore d'un sous-ensemble du type 8

par exemple : SET Z = BUTTON(2)

Type 11 : Appel de sous-programme avec retour d'erreur NODAL

par exemple : CALL SUBR(A,1+B,3,"TEXT")

Jusqu'à 8 paramètres d'usage général comme dans le type 8.

Si une erreur est détectée, le sous-programme peut provoquer l'arrêt de NODAL avec impression d'un message d'erreur (cette ressource est également disponible avec les types précédents).

Type 12 : Appel de sous-programme sans retour d'erreur NODAL

Analogue au type 11 mais sans possibilité de déclenchement d'un message d'erreur NODAL. Ce type a été prévu pour l'appel de sous-programmes déjà existants, écrits en langage d'assemblage ou en Fortran.

Type 18 : Sous-programmes libres

Il peut s'agir de fonctions de lecture/écriture ou de sous-programmes CALL. Ce sont essentiellement des sous-programmes NODAL; ils doivent utiliser les ressources internes de NODAL afin d'obtenir leur espace de travail, leurs paramètres, etc.

Types 22, 23, 24 : Fonctions de chaînes de caractères

Elles sont discutées plus en détail dans le chapitre consacré aux ressources pour la manipulation des chaînes de caractères.

MODULES A USAGE GENERAL

Le type 6 est spécialisé pour l'appel des sous-programmes de la bibliothèque FORTRAN; il n'est pas décrit en détail ici. Ce type est très important et il est décrit complètement au chapitre 7. Les types 18, 22, 23, 24 sont décrits ultérieurement. Dans le reste de ce chapitre il est discuté des types 8, 9, 10, 11, 12, à usage général, compatibles avec une rédaction en FORTRAN.

6.1 Transmission de paramètres - Appel standard en SINTRAN

NODAL effectue l'appel d'un sous-programme écrit en code machine selon la modalité standard de SINTRAN, c'est-à-dire en utilisant le registre A pour pointer vers une liste des adresses des paramètres. Cela assure la compatibilité avec des sous-programmes existants en SINTRAN, ainsi qu'avec le FORTRAN. Les paramètres fournis par NODAL au sous-programme comprennent des paramètres "explicites" contenus dans le programme NODAL ainsi que certains "paramètres implicites". Dans les types 8, 9 et 10, il est fourni deux paramètres implicites qui précèdent les paramètres explicites, par exemple dans

```
SET CAMAC(1,2,3,4) = 5
```

il y a 6 paramètres. Le premier est le paramètre VALUE, (valeur) un paramètre écrit en virgule flottante, comprenant trois mots, qui aura la valeur 5 dans l'exemple ci-dessus. Le second est le drapeau, qui est un entier ayant la valeur -1 dans l'exemple ci-dessus, ce qui signifie que cette fonction de lecture-écriture est utilisée dans le mode écriture. Il aurait la valeur +1 dans le mode lecture, par exemple dans :

```
SET Z = CAMAC(1,2,3,4)
```

Dans ce cas VALUE aura une valeur indéterminée au moment de l'entrée dans le sous-programme, mais devra contenir le résultat de la lecture à la sortie. Lors de cette sortie le sous-programme doit mettre à zéro le drapeau FLAG si tout est correct ou inversement lui affecter un numéro d'erreur. Dans ces conditions NODAL s'arrêtera et imprimera un message d'erreur. NODAL dispose d'un certain nombre de messages d'erreur "en conserve" dont la liste est donnée à l'Annexe 5. L'emploi

du numéro d'erreur correspondant entraîne l'impression du message d'erreur, par exemple :

ARGUMENT LIST ERROR AT LINE 1.10

Si le numéro d'erreur est plus élevé que ceux qui correspondent aux messages "en conserve", NODAL affiche ce numéro, par exemple,....

Les "paramètres implicites" sont alors suivis des paramètres explicites, soit 4 valeurs entières dans l'exemple ci-dessus.

Dans le type 11 de sous-programmes CALL, il n'est fourni qu'un seul paramètre implicite, le drapeau, FLAG. Il est mis à zéro à l'entrée, et à la sortie il a l'effet décrit ci-dessus.

Dans le type 12 aucun paramètre implicite n'est fourni de sorte que la liste de paramètres pointée par le registre A correspond exactement aux "paramètres explicites".

6.2. Espace de travail

L'une des caractéristiques du système NODAL-SINTRAN des NORD-10 réside dans l'emploi d'un code rentrant. Les sous-programmes NODAL doivent en effet être rentrants afin de pouvoir être utilisés à plusieurs niveaux. Tout espace de travail exigé par ces sous-programmes doit être obtenu par l'intermédiaire du système.

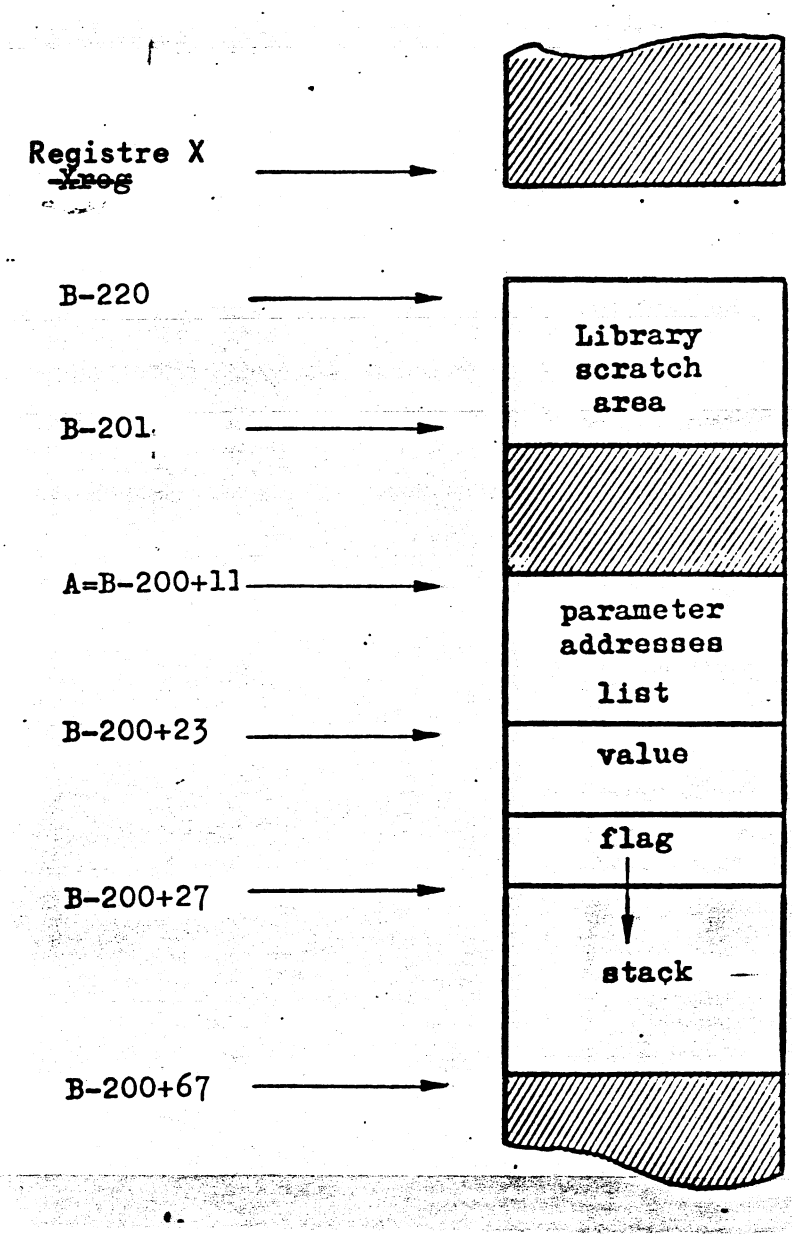
Au niveau le plus bas cet espace peut être obtenu à l'aide de SINTRAN en utilisant les sous-programmes PUSH et POP. C'est ce qui se passe

normalement lorsque les sous-programmes sont appelés à partir de programmes ou d'autres sous-programmes écrits en language d'assemblage. Toutefois l'emploi de PUSH entraîne le risque de faire passer le programme dans l'"état attente" lorsqu'il est en cours d'exécution alors qu'un autre programme de priorité plus élevée se trouve lui-même en "état attente".

Des sous-programmes appelés seulement à partir d'éléments en FORTRAN ou NODAL peuvent utiliser la zone de travail de la bibliothèque située entre les psoitions B-220 et B-201 (en octal).

Des sous-programmes ne pouvant être appelés que par NODAL peuvent utiliser d'autres ressources qui sont schématisées sur la figure ci-après.

Disposition de la pile lors d'un appel de NODAL



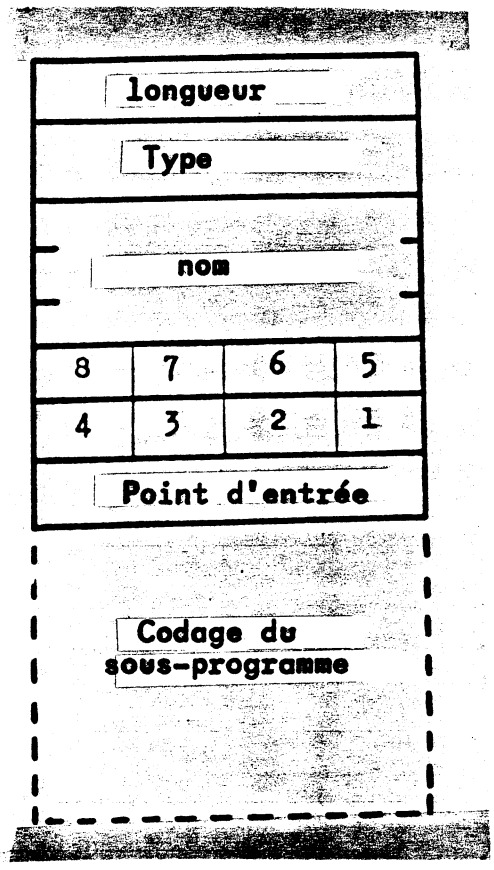
En plus de la zone de travail de la bibliothèque située entre B-220 et B-201, NODAL offre un espace de travail complémentaire. Le registre X pointe vers le dernier mot du tampon de la tâche actuellement utilisé, de telle sorte que tout l'espace compris entre cette implantation et B-201 est libre. NODAL utilise également une zone de pile de 40 mots (en octal) pour stocker des paramètres de valeur (jusqu'à 8 descripteurs de chaînes formés chacun de quatre mots). Cet espace est rarement utilisé en totalité, de sorte que la partie vide peut être employée par l'utilisateur.

Dans les sous-programmes NODAL on peut avoir accès directement aux adresses des paramètres par l'intermédiaire du registre B. On peut également avoir un accès direct aux paramètres de valeur en utilisant ce même registre car chaque paramètre de valeur est inséré successivement dans la pile, à l'intérieur de la zone de pile présentée sur la figure. Dans l'Annexe 3 on donne un exemple de sous-programme utilisant ces ressources.

Enchaînement avec NODAL - L'en-tête

Afin d'être en mesure d'appeler le sous-programme, il faut que NODAL dispose d'une certaine information représentant par exemple le nom le type, les spécifications de paramètres et l'adresse du début. Ces divers renseignements sont contenus dans l'en-tête ("HEADER") qui a la forme illustrée ci-dessous:

Disposition de l'en-tête (HEADER)



Descripteurs de paramètres

Les HEADERs NODAL doivent être inclus dans une certaine liste repérée par un nom NODAL qui peut également contenir d'autres éléments NODAL identifiés par un nom (par exemple: tableaux de données (DATA), sous programmes NODAL, etc.). Chaque enregistrement dans une liste identifiée doit commencer par une information de la longueur correspondante, étant donné que cette information servira de pointeur pour le bloc suivant qui est accolé au précédent.

La longueur doit être un nombre positif et la liste doit se terminer par -1. Il est souvent commode d'implanter le corps du sous-programme dans la zone, avec l'en-tête, selon la disposition correspondant à la ligne en pointillés tracée sur la figure. Pour les sous-programmes en code machine qui sont discutés ici, les types concernés seront les types 8, 9, 10, 11 ou 12.

Le nom (NAME) comprend 6 octets consécutifs; les octets non utilisés sont mis à zéro. Le point d'entrée représente l'adresse de début du sous-programme.

6.3. Descripteurs et types de paramètres

Dans l'en-tête on peut spécifier jusqu' huit paramètres à l'aide des deux mots descripteurs de paramètres. Ces descripteurs sont des champs de quatre bits, disposés comme le montre la figure. Un champ contenant zéro signale la fin des descripteurs. On peut donc ainsi spécifier quinze types différents de paramètres, mais actuellement neuf d'entre eux seulement sont implantés. Ils sont indiqués dans le tableau ci dessous. Le nombre de gauche représente la valeur qui doit figurer dans le champ de quatre bits pour spécifier le type de paramètre considéré.

TYPES DE PARAMETRES

1	Valeur réelle	- nombre de 3 mots, écrit en virgule flottante implanté dans la pile
2	valeur entière	- nombre entier d'un mot implanté dans la pile
3	valeur de chaîne	- descripteur de chaîne de 4 mots, implanté dans la pile
4	référence NODAL	- élément de données NODAL, adresse du premier mot, par exemple longueur
5	référence réelle	- référence de 3 mots en virgule flottante
6	référence entière	- référence d'un mot, en notation entière
7	tableau de valeurs réelles	- 3 mots par éléments du tableau, écrit en virgule flottante
8	tableau de valeurs entières	- 1 mot par élément du tableau, en notation entière
9	nom NODAL	- élément de 3 mots implanté dans la pile

REMARQUE L'adresse du paramètre est contenue dans la liste commençant à B-200+11 qui pointée par le registre A. Le paramètre lui-même est stocké ailleurs, mais les paramètres de valeur sont contenus dans la pile des paramètres, comme il en a été discuté antérieurement.

6.4 Séquences d'appel

Dans le cas de paramètres ayant des valeurs réelles et entières, on peut utiliser une expression pour définir le paramètre, par exemple:

```
SET CAMAC(SIN(PIE/Z),N,0,16) = 5
```

pourrait fournir quatre paramètres à valeur entière.

Tous les autres paramètres doivent être des noms simples, par exemple:

```
CALL FFT(A,-1)
```

instruction dans laquelle le premier paramètre est un nom de tableau.

Logiquement des éléments de tableau devraient être utilisables en réponse aux spécifications 5 et 6 mais cette possibilité n'est pas introduite actuellement. Le type 6 est utilisé lorsque le sous-programme souhaite renvoyer un entier au programme appelant. Toutefois NODAL ne travaille qu'avec des variables réelles simples, de sorte qu'une conversion automatique est effectuée à l'entrée et à la sortie du sous-programme. Actuellement cette conversion fonctionnera mal si dans le même sous-programme l'on a utilisé le même nom en réponse à plusieurs spécifications de type 6.

NOTE Un EXIT (sortie) normal doit être effectué par l'intermédiaire du registre L, à la fin du sous-programme. Le registre B doit avoir la même valeur qu'à l'entrée.

7. MODULES DE DONNEES

Les notions de variables du système et de modules de données ont été décrites dans le document LAB II-CO/Int/00/73-8. Ce chapitre décrit en détail les règles pour l'appel de modules de données et pour leur chaînage avec NODAL en utilisant une entrée identifiée de type 7 figurant dans une liste d'éléments NODAL.

7.1 Séquence d'appel en langage d'assemblage

Les modules de données sont appelés selon la modalité standard de SINTRAN. A l'entrée, le registre A pointe vers une liste d'adresses de paramètres et les sous-programmes peuvent utiliser une zone de travail comprise entre B-220 et B-201.

Quatre paramètres sont utilisés; ce sont les suivants :

a. VALUE (valeur) (trois mots en virgule flottante). Ce paramètre contient les données qui doivent être transmises à l'équipement pour affichage ou retournées après lecture.

b. FLAG (drapeau) (entier). Ce paramètre indique s'il s'agit d'une opération de lecture ou d'écriture. NODAL fournit +1 pour une opération de lecture et -1 pour une écriture. Il a été suggéré que d'autres valeurs, par exemple +2 et -2, pourraient être utilisés par des programmes en code machine déjà testés, afin de court-circuiter certaines vérifications de routine. Au retour FLAG doit être mis à zéro si tout est correct; dans le cas contraire il lui est affecté un numéro d'erreur.

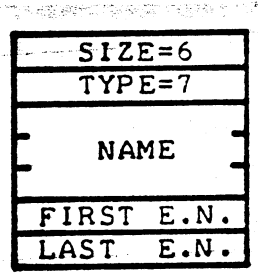
c. EQUIPMENT NUMBER (numéro d'équipement) (entier). Ce paramètre indique l'équipement particulier considéré. Le format du numéro de l'équipement est décrit dans la première référence présentée ci-dessus.

d. PROPERTY (caractéristique) (entier). Ce paramètre indique quelle caractéristique de l'équipement considéré doit être lue ou écrite.

Le module de données doit normalement vérifier la légalité des paramètres. Dans de nombreux cas il souhaitera également contrôler la priorité d'accès du programme appelant, c'est-à-dire le mot de passe de l'utilisateur. Cet élément n'est pas transmis sous la forme d'un paramètre explicite, mais il peut être obtenu par appel du sous-programme "PASWD" qui renverra le mot de passe de l'utilisateur.

7.2. Communications avec NODAL

Ces communications sont assurées par un enregistrement du type 7 affecté d'un nom figurant dans une liste d'utilisateurs de NODAL, ainsi que par un tableau de 64 mots contenant les points d'entrée des modules de données. L'en-tête est présenté ci-dessous



LONGUEUR=6
TYPE=7

NOM

PREMIER NUMERO DE L'ENREGISTREMENT
DERNIER NUMERO DE L'ENREGISTREMENT

Ainsi 7 mots sont nécessaires pour tout nom qui pointe vers un module de données. Le numéro de l'équipement comprend 6 bits indiquant son type. Ils sont utilisés par NODAL pour effectuer un branchement sur le sous-programme correct, par l'intermédiaire du tableau SVTAB résidant en mémoire. Les informations peuvent se présenter comme suit :

```
SVTAB, EQT0
       EQT1
       --
       --
       EQT63
```

où EQT0, -- EQT63 peuvent être les points d'entrée des 64 modules de données. Pour les ordinateurs équipés d'un tambour magnétique, on utilise un second tableau appelé CTLAB. Il contient le numéro de l'emplacement en mémoire dans lequel réside le module.

Le numéro de série figurant dans le numéro de l'équipement identifie l'organe particulier spécifié par ce nom. Si plusieurs équipements ont le même nom, par exemple : VPS(1), -- VPS(50), les numéros de la série correspondent à la première et à la dernière des unités désignées par ce nom.

7.3. Séquence d'appel de NODAL

Les séquences d'appel fondamentales, par exemple pour une variable du système MBBQ, sont de la forme :

```
SET MBBQ(1,#CUR)=456
SET A=MBBQ(1,#CUR)
```

Dans le premier cas, le sous-programme est appelé par le sous-programme d'affectation de NODAL; dans le second cas il est appelé par le sous-programme d'évaluation de l'expression. Ainsi la valeur lue pour la variable du système peut être utilisée dans toute expression NODAL, par exemple dans les ordres TYPE, IF, FOR.

Les spécifications concernant la caractéristique de l'équipement peuvent être omises; par exemple, l'ordre :

SET MBBQ(1)=456

sera accepté par NODAL et un numéro de caractéristique égal à zéro sera affecté au module de données. Ces modules interprètent une spécification de propriété valant zéro comme étant la caractéristique la plus couramment requise, par exemple l'intensité de courant pour une alimentation.

Le numéro de l'organe peut également être omis lorsqu'il n'est pas requis, par exemple :

SET MBBH=456

est un ordre qui reste significatif lorsqu'il n'existe qu'un seul MBBH. Le numéro de l'équipement qui est transmis est identique à celui qui figure dans l'en-tête pour le nom MBBH.

7.4 Un exemple

L'appendice 4 contient une liste d'utilisation du code source en langage d'assemblage pour un exemple de configuration d'un sous-programme du système. Les lignes 10, 11 et 12 spécifient trois en-têtes, MBBH,

MBBV et MBBQ. Ces en-têtes sont contenus dans une certaine liste NODAL d'éléments identifiés dont l'élément terminal, -1, est présenté à la ligne 12. (Les listes NODAL font l'objet d'une discussion complémentaire au chapitre 11). Dans ces éléments du type 7, le numéro de l'équipement comprend un paramètre du type 2, avec des numéros de série 100 (144 en octal), 101, 102.

L'adresse de début du module de données 2 est trouvée par NODAL comme étant le troisième enregistrement dans la table d'adresses SVTAB, soit ici : EQT2. Le reste de l'exemple présente le module de données lui même.

Le sous-programme exige deux variables de travail, SERNO pour stocker le numéro de série et RETAD pour stocker l'adresse de retour. Ces variables sont définies aux lignes 26 et 27, avec référence à la nouvelle valeur du registre B décalée de -20 à la ligne 32. En premier lieu, le numéro de l'équipement est vérifié à la ligne 31 pour contrôler qu'il est bien du type 2. Cela n'est pas nécessaire lorsque l'appel provient de NODAL, mais cela peut être intéressant lorsque le sous-programme est appelé à partir de programmes en FORTRAN ou en langage d'assemblage. Puis, dans les lignes 32 et 33 le numéro de série est vérifié pour constater s'il se situe dans la gamme qui lui est affectée. Dans le cas contraire le numéro d'erreur 35 (ILLEGAL EQUIPMENT NUMBER = numéro d'équipement illégal) est affiché. Ensuite, la légalité de la caractéristique est vérifiée; si elle est incorrecte le numéro d'erreur 36 (ILLEGAL PROPERTY = caractéristique illégale) est affiché. Si PROPERTY = 0, le programme utilise une propriété par défaut, ici : 42. Aucune vérification du mot de passe n'est effectuée ici. Finalement, le drapeau (FLAG) est vérifié et l'opération de lecture ou d'écriture appropriée est effectuée. Ici la lecture ou l'écriture ne concernent que le simple tableau de données DTB2 défini à la ligne 50.

cg/

8. FONCTIONS ET SOUS-PROGRAMMES DEFINIS EN NODAL

Le fonctionnement de NODAL est basé sur son aptitude à appeler des fonctions et des sous-programmes identifiés par un nom. Jusqu'à maintenant il n'a été discuté que de sous-programmes rédigés en langage d'assemblage. Ce chapitre montre comment des procédures identifiées peuvent être écrites en NODAL.

Dans de nombreux langages conçus pour des systèmes avec compilation et exécution, les procédures identifiées apparaissent en tant que parties dans la liste des instructions du programme principal. Il n'en est pas de même en NODAL ou les procédures peuvent être utilisées même en l'absence de programme, c'est-à-dire dans le cadre d'un ordre immédiat. Ainsi définie une procédure représente une opération isolée par rapport à la rédaction d'un programme principal. Les chapitres 6 et 7 ont montré comment écrire des procédures en langage d'assemblage; le présent chapitre en fait de même pour les procédures définies en NODAL. Une fois définies, toutes les procédures font partie du système et deviennent ainsi des éléments externes par rapport au programme principal.

Dans l'ordinateur local il est envisagé que la plupart des procédures, par exemple des modules de données, soient écrites en langage d'assemblage afin d'assurer un lien entre l'ordinateur et l'équipement. Toutefois, dans un ordinateur de pupitre les liens avec l'équipement sont assurés par l'envoi d'ordres ou de programmes en NODAL sur les lignes de transmission de données, avec retour des résultats sous contrôle d'un ordre REMIT. Sur ce dernier ordinateur, afin de simuler l'effet d'ordres immédiats ou de programmes simples on peut définir des procédures identifié NODAL comprenant des ordres EXECUTE et REMIT. Par exemple le

sous-programme BCT résidant dans l'ordinateur 5 peut être simulé dans l'ordinateur de pupitre en frappant au clavier la séquence ci-dessous :

```
1.1  EXECUTE(5) 1.2  ;  WAIT(5)  ;  VALUE X
1.2  SET X = BCT(3)  ;  REMIT X
DEFINE-FUNCTION BCTR3
```

L'ordre immédiat "DEFINE FUNCTION BCTR3" ci-dessus crée une fonction définie en NODAL appelée BCTR3. Le corps de la fonction est formé du texte présent dans le tampon au moment où est donné l'ordre DEFINE, c'est-à-dire ici les lignes 1.1 et 1.2. La valeur de la fonction est donnée par l'expression figurant dans l'ordre VALUE. Ce même ordre achève également l'exécution de la fonction, étant donné qu'il contient un ordre END.

8.1. L'ordre DEFINE

L'ordre DEFINE convertit le contenu du tampon renfermant le texte en une procédure identifiée par un nom. Trois types de procédures peuvent être créés : des fonctions numériques de lecture-écriture, des sous-programmes CALL et des fonctions d'écriture-lecture de chaînes de caractères. Elles se différencient par leurs syntaxes respectives :

```
DEFINE-FUNCTION NAME
DEFINE-CALL NAME
DEFINE-STRING NAME
```

La procédure peut contenir des paramètres, par exemple :

```
DEFINE-CALL NAME(V-ABC, R-DEF, S-GHI)
```

Dans la ligne précédente, les noms des paramètres sont ABC, DEF, GHI. Les préfixes "V-", "R-" et "S-" représentent les types de paramètres formels, soit : value(valeur), reference (référence) ou string (chaîne de caractères). Ces paramètres formels sont définis à partir de la séquence d'appel, lorsque le sous-programm est activé; ils peuvent donc être utilisés dans le corps du sous-programme. Par exemple, la fonction simple SUM peut être définie comme suit :

```
1.1  VALUE  ABC+DEF
DEF-FUN SUM(V-ABC, V-DEF)
```

et utilise entre autres dans une instruction "TYPE SUM(2,3)" qui afficherait le résultat "5". Dans certains cas des paramètres additionnels sont créés par le système. En particulier pour DEF-FUN et DEF-STR, une variable numérique supplémentaire, simple, FLAG, est toujours produite. Elle a la valeur +1 si la fonction a été utilisée dans le mode lecture et -1, pour le mode écriture. De même, si ces types sont utilisés dans le mode écriture la variable "VALUE" est créée. Il s'agit d'une variable numérique dans le cas du type DEF-FUN et d'une variable de chaîne pour un type DEF-STR. VALUE prend la valeur de l'expression figurant à droite du signe égal dans l'ordre SET, par exemple :

```
1.1  EXECUTE (8) 1.2 VALUE ; END
1.2  SET MBBH(2) = VALUE
DEF-FUN INJ:VB
SET INJ:VG = 200
```

Le premier ordre immédiat crée la procédure INJ:VB. Le second l'active avec la variable VALUE mise à 200.

Lorsque l'on utilise une fonction définie du type:

DEF-FUN NAME

deux paramètres supplémentaires, PAR1 et PAR2, sont créés. Leurs valeurs sont établies selon l'algorithme utilisé pour les modules de données qui a été décrit au chapitre 7. La séquence d'appel peut à volonté contenir zéro, un ou deux paramètres, par exemple

```
                TYPE NAME
ou              TYPE NAME(2)
ou              TYPE NAME(2,3)
```

avec PAR1=PAR2=0, ou PAR1=2, PAR2=0, ou encore PAR1=2 et PAR2=3.

Si l'on souhaite vraiment pas de paramètres, on imposera cette omission au système en écrivant :

DEF-FUN NAME( )

8.2 L'ordre OPEN

Cet ordre prend le texte de la procédure et la place dans le tampon prévu pour le texte. La procédure est éliminée. Il s'agit donc d'un ordre ayant un effet opposé à celui de l'ordre DEFINE, et il est utilisé pour l'édition, par exemple :

OPEN NAME

8.3 Les ordres VALUE et $VALUE, la fonction ERROR

L'ordre VALUE établit la valeur de la fonction lorssqu'elle est utilisée dans le mode lecture. L'ordre $VALUE en fait de même pour une fonction de chaîne de caractères. Ces deux ordres comprennent un ordre END de sorte qu'ils entraînent une sortie hors de la fonction. Pour les fonctions utilisées en mode écriture ou pour les sous-programmes CALL, un ordre END doit être utilisé. Voici deux exemples de ces ordres :

```
1.9    VALUE X+Z*Y
1.9    $VALUE "THIS THE RESULT STRING"
```

Si une erreur est rencontrée, le mécanisme standard d'affichage d'erreur NODAL peut être activé en utilisant l'instruction

```
SET ERROR = X
```

où X est le numéro de l'erreur. Cela provoque le passage de NODAL dans son mode d'affichage d'erreur, comme cela a été décrit au chapitre 2, tout comme si le sous-programme considéré était un sous-programme écrit en code machine, conformément aux description des chapitres 6 et 7.

8.4. Fonctions utilitaires

Les sous-programmes ci-après du type CALL, conçus pour l'utilisateur, sont disponibles pour la manipulation de fonctions définies :

```
LISD
SDEF FILENAME
LDEF FILENAME
ZDEF
```

LISD provoque l'affichage d'une liste des fonctions définies actuellement utilisées, en donnant leur longueur ainsi que les spécifications des paramètres.

SDEF entraîne le stockage des fonctions définies actuellement utilisées dans le fichier FILENAME

LDEF entraîne le chargement pour utilisation des fonctions conservées dans FILENAME

ZDEF élimine la liste de fonctions définies actuellement utilisées

8.5. Récursivité

Les fonctions définies en NODAL peuvent être utilisées de manière récursive. Voici un exemple :

```
1.1  IF PAR2>PAR1; VALUE HCF(PAR2,PAR1)
1.2  IF PAR2=0; VALUE PAR1
1.3  VALUE HCF(PAR2,MOD(PAR1,PAR2))
DEF-FUN HCF
```

Cet exemple crée une fonction récursive calculant le facteur commun le plus élevé de deux nombres(PGCD). En NODAL, la profondeur de récursivité est limitée par l'espace de travail disponible. Il est utilisé environ 70 mots par niveau.

Dans les langages du type à compilateur, les techniques de récursivité ne sont pas habituellement recommandées (lorsqu' elles sont possibles) du fait de leur lente exécution. Il n'en est pas de même en NODAL, lorsque le programme récursif est plus court que le programme non récursif, comme cela est souvent le cas.

9. RESSOURCES POUR LA MANIPULATION DES CHAINES DE CARACTERES

NODAL n'est pas seulement un langage dialogique ; il est également un langage de programmation des interactions, tout au moins au niveau des ordinateurs de pupitre. Pour cette application la possibilité de manipuler des chaînes de caractères est très importante. Les ressources conçues à cet effet et offertes par NODAL se divisent en deux groupes. En premier lieu, on trouve des ressources pour la manipulation des chaînes de caractères. Elles permettent la création, l'enchaînement, la subdivision, la comparaison, l'entrée et la sortie de chaînes de caractères. Ces ressources sont disponibles dans les ordinateurs de pupitre. En second lieu, il existe des ressources pour le traitement des chaînes de caractères. Elles permettent de les traiter par recherche de configurations, remplacement de sous-chaînes par d'autres, et création de nouvelles chaînes à partir d'anciennes. Ces ressources ne sont disponibles que dans l'ordinateur de service. Naturellement, les variables de chaînes et les tableaux de chaînes constituent des éléments NODAL standards, pouvant ainsi être transmis à partir d'ordinateur de pupitre vers l'ordinateur de service, en vue du traitement.

9.1. L'ordre $SET, les variables et tableaux de chaînes de caractères

Les variables de chaînes peuvent être créées à l'aide de l'ordre $SET; par exemple, l'instruction :

```
$SET A = "THIS IS A STRING"
```

crée la variable de chaîne A dont le contenu est représenté par la chaîne de caractères présente entre les guillemets. Si l'o désire que la chaîne de caractères contienne des guillemets, on pourra alors utiliser des apostrophes dans l'ordre ⊄SET, afin de délimiter cette chaîne. Par exemple :

⊄SET A = "FRED'S FRIEND SAID"

⊄SET B = '"WHERE ARE YOU"'

crée deux chaînes de caractères A et B, de sorte que l'ordre TYPE :

TYPE A B

provoquera l'impression de :

FRED'S FRIEND SAID "WHERE ARE YOU"

Des tableaux de chaînes de caractères peuvent également être utilisés. Ils doivent d'abord être créés en utilisant l'ordre DIMENSION, par exemple :

DIM-STR A

créera un tableau de chaînes A. Des éléments peuvent ensuite y être insérés en utilisant l'ordre ⊄SET, par exemple :

⊄SET A(5) = "GEORGE"

Il n'est pas nécessaire que les éléments A(1) à A(4) existent ; par exemple on pourra écrire :

⊄SET A(10) = "JOE"

Le tableau de chaînes contient maintenant deux éléments A(5) et A(10). Il existe une fonction utile, FIND qui renvoie le numéro de la position d'une chaîne dans un tableau ou qui affiche -1 si

la chaîne considérée ne figure pas dans le tableau. Par exemple :

```
TYPE FIND(A$,"JOE")
```

entraînera l'impression du nombre 10.

9.2 Fonctions de chaînes de caractères

Dans les chapitres précédents nous avons longuement traité des fonctions arithmétiques fournissant une valeur numérique lorsqu'elles sont appelées, par exemple SIN(X). Il existe des fonctions équivalentes pour les chaînes de caractères, c'est-à-dire des fonctions qui fournissent une chaîne lorsqu'elles sont appelées. Par exemple :

```
$SET B = SUBS(1,5,A)
```

affecte à la variable de chaînes B le contenu de la chaîne donnée par la fonction de chaînes SUBS. Ici, SUBS renvoie les sous-chaînes 1 à 5 de la variable de chaînes A, c'est-à-dire les cinq premiers caractères de A.

Les fonctions de chaîne peuvent entraîner une non-exécution de l'ordre $SET. Par exemple, dans la ligne 3.1 :

```
3.1 $SET A = INPUT(65):3.8; DO 4
```

l'ordre $SET affecte à A la chaîne fournie par la fonction INPUT. Il s'agit normalement de la chaîne de caractères lue dans le fichier 65 jusqu'au premier retour de chariot avec avance d'une ligne. Toutefois s'il ne reste plus de caractères dans ce fichier INPUT entraîne la non-exécution de l'ordre $SET, ce qui se tradui

par un GOTO à la ligne 3.8. Si aucun message en retour pour non-exécution n'a été spécifié (" :3.8" dans l'exemple ci-dessus), le programme se poursuivra et A sera affecté d'une chaîne de longueu nulle. Ce mécanisme particulier d'incident ne doit pas être confondu avec des erreurs normales entraînant un arrêt du programme NODAL. Par exemple, à la ligne 3.1 ci-dessus il se produirait une erreur normale si le fichier 65 n'avait pas été ouvert pour lecture. Cela entraînerait l'arrêt du programme avec affichage d'un message d'erreur NODAL normal.

9.3 Enchaînements

Dans les exemples ci-dessus nous avons utilisé des chaînes de caractères et des fonctions de chaînes constantes, en vue de spécifier des chaînes. Il s'agit là de cas particuliers d'enchaînements. Un enchaînement correspond à une série d'éléments définissant des chaînes et fournissant une nouvelle chaîne comprenant la série des diverses chaînes précédentes accolées bout à bout. Par exemple :

```
$SET A=" AND GEORGE"
$SET B="JOE" SUBS(1,5,A) "FRED"
```

affecterait à la chaîne B le contenu "JOE AND FRED". Un enchaînement peut comprendre une série d'éléments de chaînes, d'expressions arithmétiques ou des caractères de contrôle. Ces derniers sont exactement les mêmes que ceux qui ont été définis au chapitr 2 pour l'ordre TYPE. Des expressions arithmétiques sont transformées en chaînes de caractères à nouveau comme dans le cas de l'ordre TYPE, en utilisant soit le format par défaut soit un format spécial, repéré par le caractère de contrôle %. Ainsi :

```
$SET SESCAN(2)="CURRENT =" %3 KNOB
```

entraînera un sondage automatique de l'unité 3, suivi de l'impression d'un message, par exemple :

CURRENT = 54

c'est-à-dire la chaîne constante "CURRENT =" suivie par la valeur de KNOB convertie en une chaîne de caractères conformément au format précédent défini par le caractère de contrôle %. Dans un enchaînement, des éléments peuvent être séparés par des espaces mais non par des virgules, comme cela est facultatif dans l'ordre TYPE. Dans les notes qui suivent, la chaîne "CONC" sera utilisée pour spécifier tout enchaînement légal, comme il vient d'être décrit ici.

9.4 L'ordre $ASK

Cet ordre permet de charger des chaînes depuis le terminal. L'ordre $ASK affiche deux points (":") puis attend que l'utilisateur frappe une suite de caractères suivie par un retour de chariot. Cette suite de caractères est ensuite affectée à la variable spécifiée dans l'ordre $ASK; par exemple :

$ASK "GOOD OR BAD" A

entraînerait l'impression de "GOOD OR BAD :" sur le télétype, puis affecterait la suite frappée au clavier à la variable A. Tout comme dans le cas de l'ordre ASK numérique, plusieurs variables, peuvent avoir une affectation demandée à l'aide d'un seul ordre $ASK.

9.5 L'ordre $IF

Cet ordre peut être utilisé pour comparer deux chaînes; par exemple :

$IF (A-"YES") 2.1, 2.2, 2.3

comparera le contenu de la variable de chaîne A avec la chaîne constante "YES" et procédera à des branchements selon le résultat de la comparaison. Il s'agit d'une comparaison lexicologique entre deux enchaînements séparés par le signe moins. Ainsi, si selon le lexique A se trouve avant "YES", le contrôle passe à la ligne 2.1 ; si A est identique à "YES", le contrôle passe à 2.2 ; si A se situe après "YES", le contrôle passe à 2.3. La rédaction du branchement conditionnel peut être abrégée comme cela a été expliqué pour l'ordre IF arithmétique.

9.6 L'ordre $DO

Le fait que NODAL soit un langage interprétatif offre une autre possibilité intéressante, mise en oeuvre par l'ordre $DO; par exemple :

```
$DO A(5)
```

prélévera le contenu de A(5), le considérera comme un ordre immédiat et l'exécutera. La chaîne de caractères est d'abord copiée dans l'ordre $DO avant l'exécution, ce qui permet de rédiger des instructions du type :

```
10.1 $DO "ERASE 10.1; DEF-F NAME"
```

Dans ce cas il y a d'abord effacement de la ligne 10-1, puis le reste du tampon est défini comme représentant la fonction NAME.

9.7 Quelques fonctions utiles

Ci-dessous nous décrivons quelques fonctions de chaînes et fonctions à valeur arithmétique, qui sont utiles dans des travaux sur les chaînes de caractères.

STRING = SUBS(I,J,CONC)	extrait la sous-chaîne I à J de l'enchaînement
VALUE = SIZE(CONC)	fournit le nombre de caractères de l'enchaînement
VALUE = EVAL(CONC)	considère CONC comme une expression arithmétique et fournit sa valeur
VALUE = FIND(ARRAY,CONC)	fournit le numéro de la position de CONC dans le tableau et donne -1 si cette chaîne n'y est pas trouvée
VALUE = OPEN(CONC1, CONC2)	ouvre pour lecture si le premier caractère de la chaîne CONC1 est "R" (pour "READ" = lire) ou pour écriture s'il s'agit d'un "W" (pour "WRITE" = écrire Ouvre le fichier dont le nom est donné par la chaîne CONC2 et fournit une valeur logique.
CALL CLOSE(VALUE)	ferme le fichier de la valeur logique VALUE. Ferme tous les fichiers si VALU ő la valeur -1.
STRING = INPUT(VALUE)	extrait une chaîne de caractères du fichier ayant la valeur logique VALUE jusqu'au premier retour de chariot ave avance d'une ligne. Elimine le retour du chariot et l'avance d'une ligne, c'est-à-dire prélève la première ligne du fichier.
STRING = INPUT	obtient une ligne prélevée sur le terminal (suppose implicitement : VALUE =
STRING = INPC(VALUE)	prélève un caractère dans un fichier.
STRING = ALPHA	variable de chaîne prédéfinie, renvoie un caractère de A à Z
STRING = NUM	variable de chaîne prédéfinie, renvoie un chiffre de 0 à 9.

Des fonctions de chaîne peuvent également apparaître à la gauche du signe égal dans un ordre $SET, par exemple :

$SET STRFUN(...) = CONC

Cette instruction est utilisée pour les dispositifs de pupitre tels que les unités d'exploration automatique, etc., dans le cadre d'un système d'exploitation en temps partagé, avec les formes :

OUTPUT(VALUE) = CONC	charge CONC dans le fichier VALUE, puis effectue un retour de chariot avec avance d'une ligne
OUTPUT = CONC	sort une ligne sur le terminal
OUTC(VALUE) = CONC	sort CONC sans retour de chariot ni avance d'une ligne.

9.8. Les ordres $MATCH et $PATTERN

Il s'agit des principaux ordres pour le traitement des chaînes de caractères, pouvant être utilisés sur l'ordinateur de service. Voici un exemple :

2.7 $MATCH STRING PATTERN : 2.1; DO 5

Cette instruction compare STRING et PATTERN. S'il y a correspondance entre ces deux chaînes, l'ordre suivant, ici : "DO 5" est exécuté. S'il n'y a pas correspondance, le champ de défaut, ici : ": 2.1", est utilisé et dans ce cas il y a branchement à la ligne 2.1. Le champ de défaut est facultatif, comme dans l'ordre $SET. Il existe une forme plus générale, bien que moins souvent utilisée par exemple :

```
2.7  $MATCH <CONC> PATTERN : 2.1
```

dans laquelle STRING a été remplacé par un enchaînement valide quelconque figurant entre deux signes d'inégalité.

Les configurations concernées (patterns) peuvent être "immédiates", c'est-à-dire qu'elles sont définies au moment de leur utilisation dans un ordre $MATCH, ou être définies et stocké avant leur utilisation par l'ordre $MATCH, par exemple :

```
$PATTERN P1 = PATTERN
```

définit et enregistre dans le fichier P1 la configuration figuran à droite du signe égal.

Ce paragraphe a décrit la syntaxe des ordres $MATCH et $PATTERN, mais il n'a pas précisé ce que l'on entend par correspondance et par configuration. Il s'agit là de la principale notion intervenant dans le traitement des chaînes de caractères; elle est décrite dans le paragraphe suivant.

9.9. Les configurations et la correspondance entre configurations

Dans le cas le plus simple, une configuration est une chaîne de caractères constante. Par exemple, la suite d'instructions :

```
2.1  $MATCH A "YES" : 2.2; TYPE "INCLUDED"; RETURN
2.2  TYPE "NOT INCLUDED"
```

représente un groupe qui va étudier le contenu de la variable de chaîne A en cherchant si la configuration "YES" y figure. Si tel est le cas, par exemple si A = "OYES" ou "OH YES" ou "YES OR NO" ou "YESSIR", le programme affiche "INCLUDED" (inclus) dans le cas contraire, il imprime "NOT INCLUDED".

Les configurations peuvent être plus compliquées que des simples chaînes de caractères. Elles peuvent inclure un enchaînement et une alternative. L'alternative est désignée par un point d'exclamation ("!"), par exemple, la configuration :

```
"YES" !"NO"
```

cherchera s'il y a correspondance soit pour "YES", soit pour "NO" la configuration

```
"A" "B" !"C" "D"
```

cherchera une correspondance soit pour la chaîne "AB", soit pour la chaîne "CD", tandis que :

```
"A" ("B" ! "C") "D"
```

recherchera la correspondance pour les chaînes "ABD" ou "ACD".

Les configurations peuvent également comprendre des champs de remplacement ou d'affectation, par exemple :

```
$MATCH A "YES" = "*"
```

recherchera la chaîne "YES" dans la configuration A; si cette chaîne y figure, elle y sera remplacée par la chaîne "*". Ainsi, si A était la chaîne "YESSIR", cette instruction la transformera en "*SIR". Le champ d'affectation affecte à une variable la partie de la chaîne pour laquelle la correspondance a pu être établie, par exemple :

```
$PAT P1 = "A" ("B" ! "C") "D"
FOR I = 1,10; $M A(I) P1 .OUTPUT
```

Le premier ordre crée une configuration P1. Le second explore le tableau A, en recherchant s'il y a correspondance entre chacun de ces éléments et P1. S'il y a effectivement correspondance, la partie concernée c'est-à-dire ici "ACD" ou ABD" est affectée à la sortie. Le point (".") est appelé opérateur d'affectation conditionnelle. (Remarque : dans l'instruction ci-dessus, le point doit être séparé de "P1" par un espace, étant donné que P1 est un nom NODAL valide qui n'est pas la configuration P1).

Les configurations peuvent être formées de plusieurs parties, par exemple :

```
$PAT P3 = P1 .A1 P2 .A2
```

Cet ordre crée une nouvelle configuration, P3, qui se compose de P1 suivi de P2. Il n'y a correspondance que si l'on peut établir une correspondance pour l'ensemble P1 immédiatement suivi de P2. Dans ce cas la partie en correspondance avec P1 est affectée à la variable A1, et la partie en correspondance avec P2 est affectée à la variable A2. S'il n'y a pas correspondance pour P1 P2, il n'y a pas affectation. L'"opérateur d'affectation immédiate" représenté par le symbole du dollar ("$") permet la copie de correspondances partielles. Par exemple, la configuration P4 :

```
$PAT P4 = P1 $ A1 P2 .A2
```

effectue la correspondance dans les mêmes conditions que P3, ci-dessus. Toutefois une telle tentative reste considérée comme réussie dans la mesure ou il y a effectivement correspondance

pour P1, alors qu'il n'y a pas correspondance pour P2. L'opérateu d'affectation immédiate, "$", intervient immédiatement dès que la correspondance pour P1 a été établie, sans attendre s'il y aura correspondance pour P2. La correspondance entre configuratic se réalise comme suit. Le programme recherche d'abord à l'établir en partant de la position du premier caractère de la chaîne explo réé. S'il n'y a pas correspondance, la tentative est renouvelée à partir de la position du second caractère, et ainsi de suite. Par exemple si A est donné par :

```
$SET A = "YES, YES, YESNO"
```

il s'ensuit que :

```
$MATCH A ("YES" $ OUTPUT "NO").OUTPUT
```

imprimera :

```
YES
YES
YES
YESNO
```

9.10. Fonctions de configurations

Dans le paragraphe précédent nous n'avons considéré que des configurations formées de chaînes de caractères. La plupart des configurations intéressantes utilisent des fonctions de configurations; par exemple :

```
"A" ARB "D"
```

une
est/configuration qui établira une correspondance pour un "A" suivi par un caractère quelconque jusqu'à "D" inclus, car ARB est une fonction de configuration établissant la correspondance pour un caractère quelconque.

Dans de nombreuses fonctions de configuration, la notion de position du curseur est une notion importante. Normalement une correspondance entre configurations commence avec le curseur en position zéro, c'est-à-dire immédiatement à gauche du premier caractère. Le programme tente d'établir la correspondance entre l chaîne et la configuration, en conmmençant avec le curseur en position zéro. S'il n'y a pas correspondance, le curseur est dépl cé jusqu'à la position 1, et la correspondance est à nouveau recherchée à partir du second caractère. Par exemple, l'ordre :

$M "OYES" "YES"

essaie d'abord de faire comprendre "OYE" avec "YES", et comme il n'y a pas concordance, le départ est décalé d'une position, ce qui amène une correspondance.

POS(N) est une fonction de configuration qui n'établit une correspondance que lorsque le curseur est en position N. Il y a donc "ancrage" de la configuration qui doit se présenter à partir d'un point déterminé et fixe dans la chaîne de caractères, se situant souvent d'ailleurs à son commencement. Par exemple :

POS(0) "YES"

ne trouve une correspondance que si "YES" apparaît au début de la chaîne.

RPOS(N) agit comme POS(N), mais la position est comptée depuis la fin de la chaîne, ce qui fait que RPOS(0) se situe immédiatement à droite du dernier caractère.

LEN(N) recherche la correspondance d'une chaîne quelconque de longueur N. Cette fonction est souvent utilisée pour vérifier la longueur d'une chaîne avant de poursuivre la recherche ultérieure de la correspondance.

SPAN(CONC) recherche la correspondance de la plus longue des suites des caractères contenues dans CONC, en commençant à la position actuelle du curseur ; par exemple :

POS(0) SPAN(ALPHA) ,LABLE

peut prélever une étiquette alphabétique formant le front d'une chaîne et l'affecter à la variable LABLE.

BREAK(CONC) a un effet inverse de celui de SPAN : il recherche la correspondance jusqu'à un caractère de rupture de suite, exclus, contenu dans CONC.

ARB recherche la correspondance pour un nombre quelconque de caractères.

ANY(CONC) recherche la correspondance avec un caractère quelconqu contenu dans CONC.

NOTANY(CONC) en fait de même pour un caractère quelconque non contenu dans CONC.

TAB(I) recherche la correspondance jusqu'à la position I.

RTAB(I) en fait de même jusqu'à la position I comptée depuis la fin de la chaîne. Par exemple :

TAB(0) RTAB(0)

recherche toujours la correspondance pour la totalité de la chaîne.

FAIL est une configuration telle que si elle est essayée cela entraîne une non-correspondance avec essai ultérieur d'autres alternatives.

ABORT est une configuration qui, lorsqu'elle est essayée, provoque l'échec complet de l'ordre $MATCH, c'est-à-dire qu'aucune autre alternative n'est essayée.

10. FONCTIONS DIVERSES

De nombreuses ressources utiles de NODAL sont rendues disponibles sous la forme de fonctions plutôt que d'ordres. Toutefois des fonctions d'appel peuvent beaucoup ressembler à des ordres étant donné que la directive "CALL" est facultative, par exemple :

```
LIST
```

fournit une liste des instructions du programme ; c'est un ordre, alors que :

```
LISV
```

donne une liste des variables et constitue une ressource disponible en tant que fonction. Toutefois deux différences peuvent être notées : en premier lieu, les ordres peuvent être écrits en abrégé alors que les appels par fonctions ne peuvent l'être ; en second lieu, la présence des fonctions est facultative dans toute configuration donnée d'ordinateurs; elles peuvent donc être omises afin de gagner de la place.

10.1 ODEV

Cette fonction provoque un déroutage de la suite de données en sortie qui au lieu d'être dirigée vers le terminal normal, se trouve acheminée vers l'organe de sortie spécifié ; par exemple, dans un système en temps partagé, la suite d'instructions :

```
SET ODEV = 5; LIST
```

entraînera le listage du programme sur l'organe de sortie 5, c'est-à-dire sur l'imprimante par ligne. A la fin de l'ordre immédiat concernant une ligne ou le programme, ODEV est automatiquement ramené sur le terminal utilisé en mode dialogique.

10.2 ERROR

Lorsqu'un programme NODAL rencontre une erreur, il s'arrête et imprime un message d'erreur comme cela a été décrit au chapitre 2. Cette action peut être provoquée artificiellement en utilisant l'ordre :

SET ERROR = N

où N peut être un entier positif quelconque, supérieur à 0. Certaines valeurs de N correspondent à des messages d'erreur particuliers, maintenus en "conserve" ; ils sont imprimés comme il est décrit à l'Annexe 5. Dans le cas contraire, c'est-à-dire si N est supérieur au numéro maximal des messages d'erreur " en conserve", un message tel que :

ERROR 143 AT LINE 1.1

est imprimé pour N = 143, par exemple. Au chapitre 8, on a décrit un emploi particulier de la fonction ERROR pour des fonctions de chaîne définies en NODAL.

10.3 LISV

Cette fonction fournit la liste des variables et tableaux de l'utilisateur, en donnant les valeurs des variables et les dimensions des tableaux.

10.4 LUST

Cette fonction donne la liste des modules en code machine qui sont disponibles pour l'utilisateur, avec leurs types et leur paramètres.

10.5 HELP

Cette fonction fournit la liste des ordres NODAL disponible

10.6 BIT

Cette fonction permet de manipuler des bits isolés dans un entier de 16 bits représenté dans la variable, par exemple : A, dans des appels tels que :

```
SET BIT(1,A) = 1
SET Z = BIT(15,A)
```

avec des numéros de bits allant de 0 à 15, et A étant une variable simple.

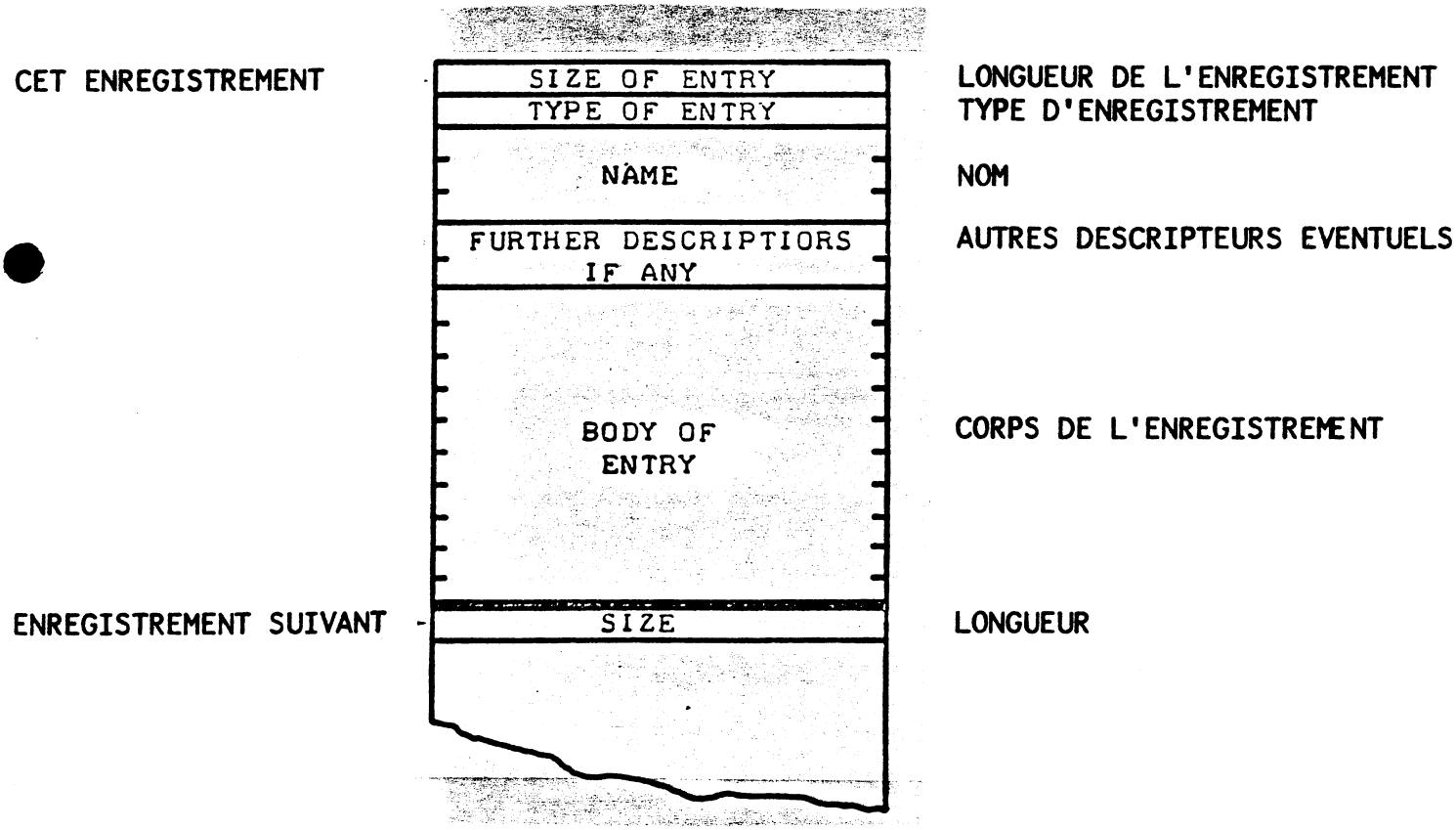
10.7 ARSIZE

Cette fonction renvoie la longueur d'un tableau en NODAL, par exemple :

```
$SET STR(ARSIZE(STR)+1) = "NEW ELEMENT"
FOR I = 1,ARSIZE(B); TYPE B(I)
```

11. LISTES D'ELEMENTS NODAL

NODAL agit sur l'accélérateur par l'intermédiaire de sous-programmes en code machine ou de liens assurant le chaînage des tableaux de données avec des tâches autonomes, rédigées également en code machine. Ces sous-programmes et tableaux de données sont contenus dans la liste des éléments NODAL. Il s'agit d'une liste chaînée, indexée par un nom, qui peut résider en mémoire, ou dans une charge mémoire, ou simultanément dans ces deux organes. Il peut être fourni plus d'une liste, en fonction de la configuration de l'ordinateur. NODAL connaît l'adresse de départ de la liste, et de ce fait lorsqu'il rencontre un nom il peut explorer la liste pour rechercher des informations complémentaires. La structure de la liste est présentée sur le schéma ci-dessous.



Les différentes parties de l'enregistrement sont utilisées co mme suit:

a) Longueur de l'enregistrement

Cet élément joue un double rôle; il indique à l'interpréteur le nombre de mots de l'enregistrement lorsque certains ordres requièrent une telle information, et il assure également le chaînage avec l'enregistrement suivant, étant donné que les divers enregistrements sont chargés bout à bout.

b) Type d'enregistrement

De nombreux types d'enregistrements peuvent être utilisés. Nous pouvons citer par exemple les variables simples, des variables de lecture seule, des tableaux, des sous-programmes pour les variables du système... Une liste des types d'éléments NODAL est présentée à l'Annexe 6.

c) Nom

Cette indication occupe trois mots pour un nom formé de six caractères.

d) Autres descripteurs

Ces éléments fournissent des informations complémentaires sur l'enregistrement, lorsque cela est requis. Par exemple, pour un sous-programme en code machine on devra trouver un mot définissant le point d'entrée ainsi qu'un mot contenant un descripteur de paramètres.

e) Corps de l'enregistrement

Cette partie contient les données ou le code. Dans le cas de sous-programmes en code machine, le code sera chargé ailleurs étant donné que le point d'entrée est donné sous une forme absolue.

11.1 Types d'éléments NODAL

Un résumé des types d'éléments est donné à l'Annexe 6. Ce paragraphe présente des informations complémentaires sur chaque type.

Type 2 - variable de lecture/écriture

Il s'agit d'un enregistrement formé de huit mots, comprenant la longueur usuelle, le type et le nom, avec en plus trois mots donnant la valeur de la variable en virgule flottante.

Type 3 - variable de lecture seule

Ce type est identique au type 2, mais NODAL ne peut pas changer la valeur ; il peut seulement la lire. PIE en constitue un exemple; cette variable est écrite comme suit (nombres en octal):

```
10; 3; #PI; #E$-##$; 0; 040002; 144417; 155242
```

Type 4 - tableau numérique de lecture/écriture

En plus de l'en-tête, un élément d'un tableau contient deux mots descripteurs. Le premier donne le nombre de lignes et le type. Le type occupe les quatre bits les plus à droite, c'est-à-dire les bits 0 à 3. Il contient 1, pour un entier, 3, pour un nombre en virgule flottante. Les 12 bits les plus à gauche donnent le nombre de lignes, de sorte que le nombre maximum de lignes est égal à 4095. Le second mot descripteur contient le nombre total d'éléments, qui est bien entendu un multiple entier du nombre de lignes. Ainsi un tableau d'entiers contenant 64 mots pourrait être décrit comme suit :

```
107; 4; #AR; #RA; #Y$-##$; 2001; 100; *+100/
```

Type 5 - Appel d'une fonction en FORTRAN

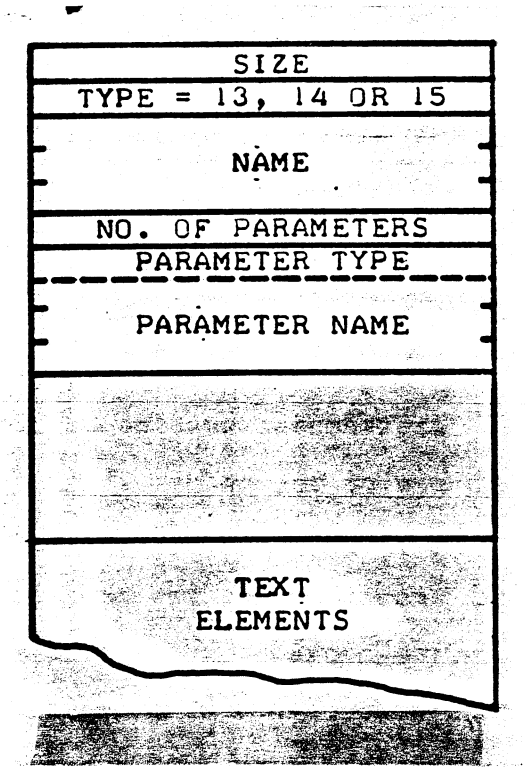
Cet enregistrement permet à NODAL d'appeler une fonction de la bibliothèque NODAL. Deux mots sont utilisés en plus de l'en-tête, soit: le nombre de paramètres et le point d'entrée du sous-programme. Il peut y avoir un ou deux paramètres. Le premier paramètre est chargé dans le TAD et le second est pointé par X. Le résultat est renvoyé au TAD, alors que X est détruit. Le sous-programme effectue un retour avec saut si tout est correct; dans le cas contraire, il effectue un retour direct avec A = numéro d'erreur. Le sous-programme peut utiliser les positions B-20 à B-1 comme zone de travail. Les fonctions de la bibliothèque NODAL énumérées au paragraphe 2.11.2 sont appelées en utilisant de tels en-têtes, mais elles peuvent également être utilisées dans des fonctions définies par l'utilisateur, conformément aux règles mentionnées ci-dessus.

Types 7 à 12 - Modules en code machine

Ils ont été décrits dans les chapitres 6 et 7.

Types 13, 14 et 15 - Fonctions définies en NODAL

Elles sont créées par l'ordre DEFINE et ne sont pas normalement employées dans une liste établie par l'utilisateur. Leur structure est présentée ci-dessous.



LONGUEUR
TYPE = 13, 14 ou 15

NOM

NOMBRE DE PARAMETRES
TYPE DE PARAMETRES

NOM DE PARAMETRES

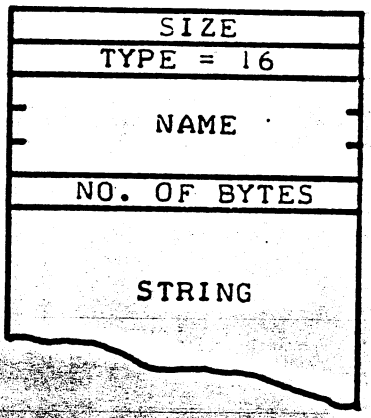
ELEMENTS DU
TEXTE

Le nombre de paramètres peut être égal à -1, 0 ou 1-8. -1 est obtenu par l'ordre DEF-FUN, sans argument; il entraîne la création des variables PAR1 et PAR2, comme il est décrit au chapitre 8. 0 signifie absence de paramètres. Les nombres 1 à 8 indiquent le nombre de paramètres requis. Pour chaque paramètre il est utilisé un bloc descripteur formé de 4 mots. Le premier donne le type de paramètre, soit: 1 pour valeur, 2 pour référence et 3 pour chaîne de caractères. Les 3 autres mots donnent le nom du paramètre formel.

Types 16 et 17 - Variables et tableaux de chaînes de caractères

Ces types d'éléments sont créés par les ordres NODAL $SET et DIM-STR. Etant donné que leur longueur et dynamique, ils ne peuvent pas être employés dans les listes des utilisateurs. Leur structure est la suivante :

VARIABLE DE CHAINE DE CARACTERES



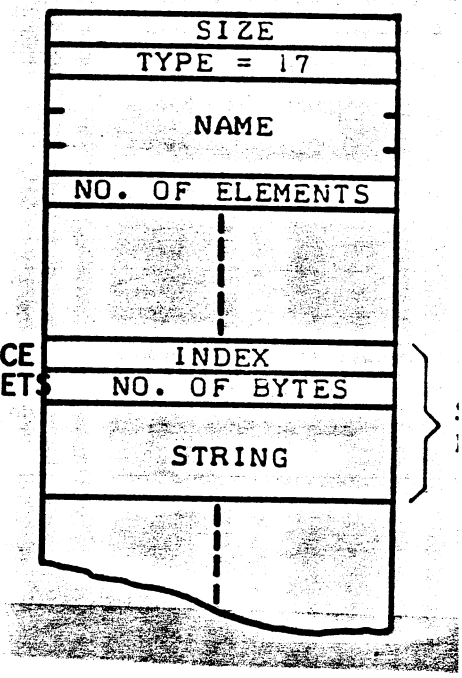
LONGUEUR
TYPE=16

NOM

NOMBRE D'OCTETS

CHAINE DE CARACTERES

TABLEAU DE CHAINES DE CARACTERES



LONGUEUR
TYPE = 17

NOM

NOMBRE D'ELEMENTS

INDICE

NOMBRE D'OCTETS

CHAINE DE CARACTERES

ELEMENT DU TABLEAU DE CHAINES DE CARACTERES

Type 18 - Sous-programme en code machine libre

Ce type d'élément confère à l'utilisateur une complète liberté dans la définition de la syntaxe de la liste de paramètres. Toutefois, il doit utiliser les sous-programmes internes de NODAL afin d'obtenir un espace de travail, des paramètres de traitement, etc. L'en-tête comprend simplement la longueur, le type, le nom ainsi que le point d'entrée du sous-programme. L'information est transmise au sous-programme comme suit:

D = DRAPEAU ; 0, pour un appel; +1 pour une lecture; -1 pour une écriture

X = POINTEUR vers des arguments de chaînes de caractères. La chaîne commence au premier caractère suivant le nom du sous-programme qui doit la lire complètement.

A = POINTEUR vers la valeur, pour une fonction d'écriture.

T = POINTEUR vers l'en-tête de l'élément

TAD = VALEUR en retour fournie par la fonction lecture.

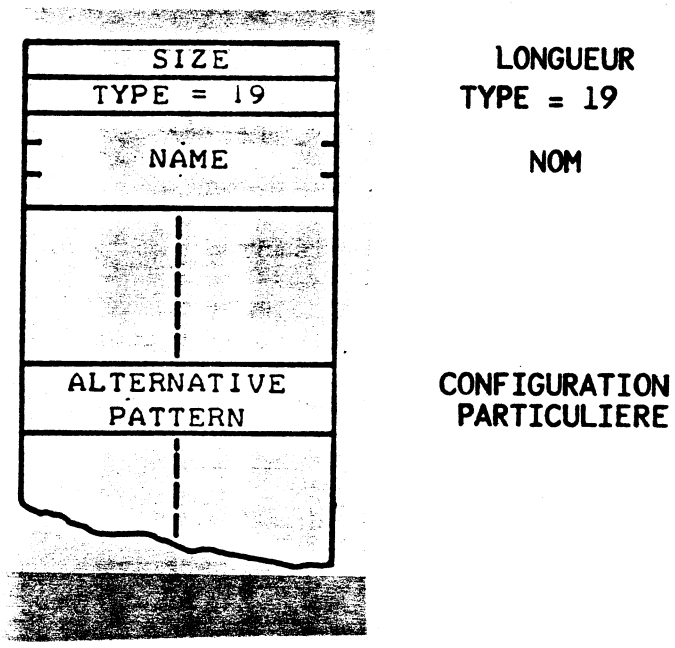
Le sous-programme doit effectuer un retour avec saut si tout est correct ou, au contraire, un retour normal avec A = NUMERO D'ERREUR.

NOTE

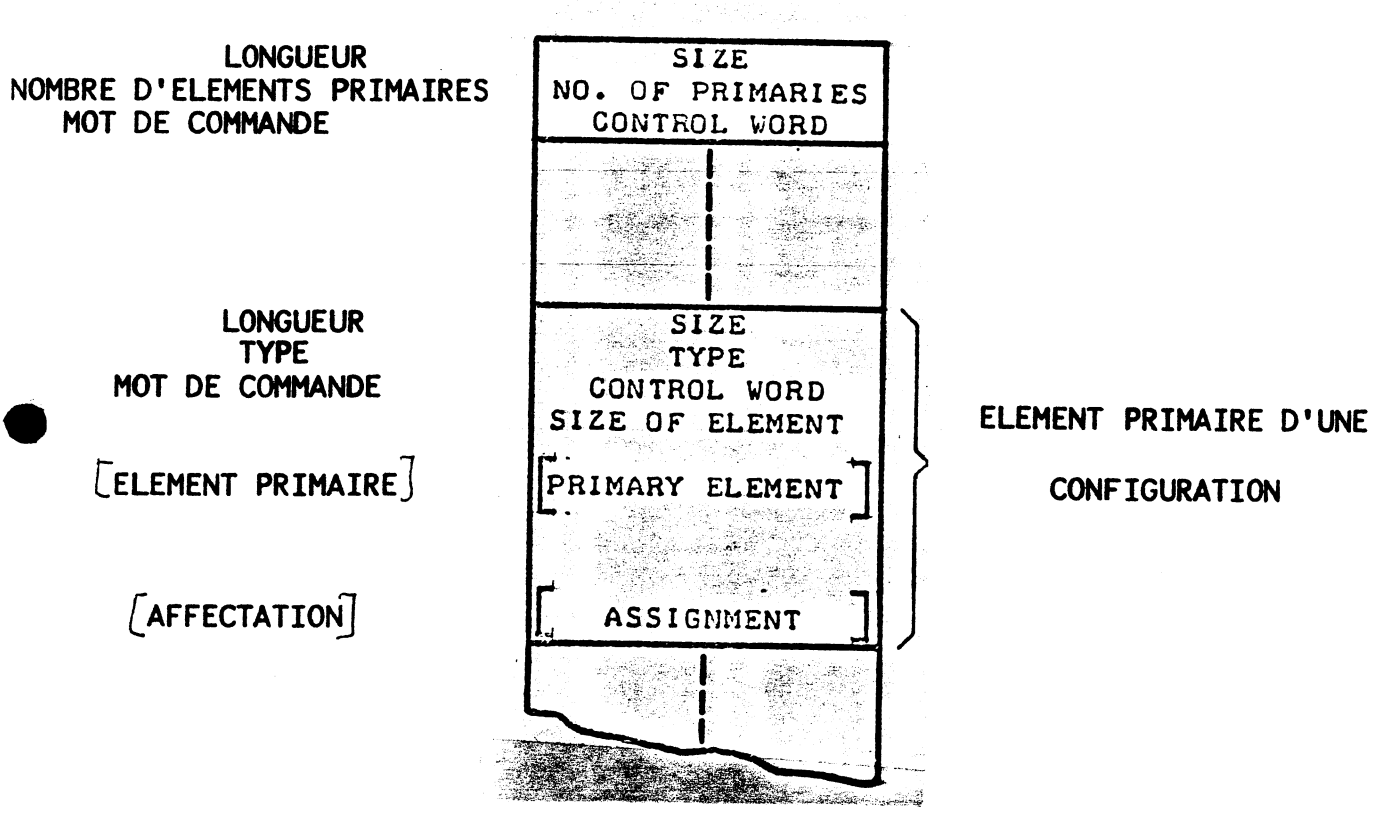
L'affectation T = ADRESSE de l'élément s'applique à tous les types 8, 9, 10, 11, 12, 18, 22, 23 et 24. Ce fait peut être utilisé pour assurer une commutation de charge mémoire, lorsque l'en-tête est résident en mémoire alors que le sous-programme est dans une autre charge mémoire.

Type 19 - Variable de configuration

La structure d'une variable de configuration est la suivante :



La structure de chaque configuration particulière est la suivante :



Type 21 - Fonction de configuration

L'en-tête d'une fonction de configuration est le même que celui d'une fonction libre du type 18, c'est-à-dire qu'il comprend la longueur, le type, le nom et le point d'entrée de la fonction. Les ressources NODAL doivent être à nouveau utilisés pour l'espace de travail et les paramètres. L'emploi du registre est le suivant :

T = chaîne de l'argument

X = chaîne du sujet

Renvoi de AD = pointeurs de lecture et d'écriture pour établir une correspondance

Retour sur échec : 1, en cas d'échec

Retouf sur échec : 2, en cas d'abandon.

Types 22, 23 et 24 - Fonctions de chaînes de caractères

Elles sont analogues aux fonctions libres de type 18, mais avec l'emploi du registre conforme à ce qui suit :

D = DRAPEAU pour fonctions de lecture/écriture
+1 pour lecture, -1 pour écriture

X = pointeur vers la chaîne de l'argument

A = pointeur vers la chaîne obtenue en résultat, pour lecture, chaîne devant être sortie pour écriture

T = pointeur vers l'en-tête de l'élément.

Retour avec saut en cas de retour normal; retour sur échec avec A = NUMERO D'ERREUR; numéro d'erreur spécial, SERR2, indiquant un échec de saut requis pour un ordre $SET.

Type 25 - Pointeur de référence

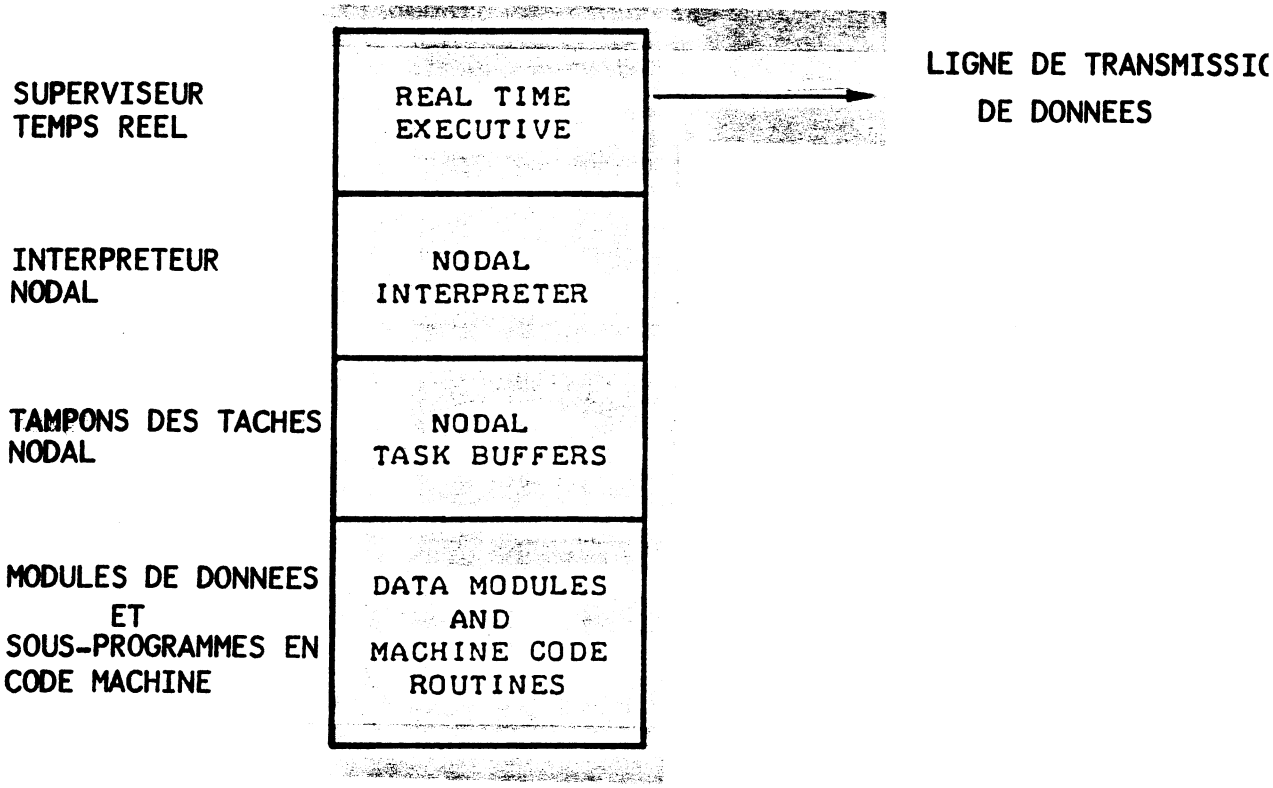
Créé temporairement au moment de l'exécution et figurant dans une fonction définie pour pointer vers une variable de référence.

12. ORGANISATION DU SYSTEME NODAL

Ce chapitre décrit comment le système NODAL est organisé dans un ordinateur de type donné. Quatre exemples sont analysés : un système simple utilisant uniquement la mémoire centrale, un ordinateur de pupitre, un ordinateur polyvalent et un ordinateur affecté à une zone d'expérimentation. La description de l'organisation précise l'implantation en mémoire ainsi que la fonction de chacune des parties de la mémoire.

12.1 Système simple utilisant uniquement la mémoire centrale

L'implantation en mémoire d'un tel système est la suivante :



Les fonctions des différentes parties sont les suivantes :

a) SUPERVISEUR TEMPS REEL

Cette partie comprend le système d'exploitation de base ainsi que des "additifs". Le superviseur gère les entrées et sorties des périphériques de l'ordinateur, les modules CAMAC et le multiplexeur, ainsi que la ligne de transmission de données. Il organise également les tâches en tenant compte des priorités et du temps.

b) INTERPRETEUR NODAL

Il s'agit d'une partie rentrante du code machine entraînant l'exécution d'un programme NODAL. Un programme NODAL particulier est formé pour l'essentiel d'une chaîne de caractères, avec des données que l'interpréteur convertit en action à entreprendre.

c) TAMPONS DES TACHES NODAL

Ces mémoires-tampon contiennent le programme NODAL en cours d'interprétation, ses variables ainsi que la pile affectée à l'éxécution qui est utilisée par l'interpréteur. Une tâche NODAL occupe un tampon en continu, depuis son lancement jusqu'à l'instant de son achèvement. Toutefois le programme réellement présent dans le tampon peut changer, en réponse à des ordres RUN, LOAD ou à des ordres similaires. Etant donné que l'interpréteur NODAL est rentrant, il peut y avoir autant de tâches NODAL "simultanément" actives qu'il y a de tampons affectés aux tâches. Dans ce système utilisant seulement la mémoire centrale, tous les tampons doivent se trouver dans cette mémoire, de sorte qu'il n'est proposé que trois tampons.

Le tampon dialogique est utilisé pour la programmation dialogique

normale. Il est attaché au terminal et tout programme frappé au clavier est stocké dans ce tampon. Par ailleurs, tout programme RUN issu d'un fichier est transféré dans ce tampon pour exécution. S'il n'est prévu qu'un tampon de ce type, il s'ensuit qu'à un instant donné un seul terminal peut être actif.

Le tampon temps réel est utilisé pour des programmes transmis par d'autres ordinateurs par l'intermédiaire de l'ordre EXECUTE. Il est également employé pour des programmes chaînés à des interruptions ou ordonnancés pour exécution à des instants déterminés. La philosophie de l'emploi de ce tampon reflète le fait que de nombreux programmes devront être exécutés au cours d'un cycle donné de l'accélérateur. Aussi de tels programmes doivent-ils être courts et susceptibles d'être traités directement jusqu'à achèvement. En particulier, l'ordre WAIT ne doit pas être utilisé. Normalement des ordres WAIT doivent intervenir au niveau dialogique, par exemple, sur l'ordinateur de pupitre, alors que des tâches en temps réel ne doivent être transmises sur les lignes de transmission de données qu'à l'instant spécifié.

Le tampon d'ordre immédiat est une petite zone tampon conçue pour la gestion des ordres immédiats ayant une priorité élevée, habituellement créés à distance en utilisant l'ordre IMEX. Ce tampon sera effectivement utilisé en cours d'exploitation, occasionnellement pour interroger directement un certain module de données éloigné, mais le plus souvent pour interroger et connaître l'état d'un ordinateur à distance ou pour abandonner une certaine tâche qui s'est mise en boucle.

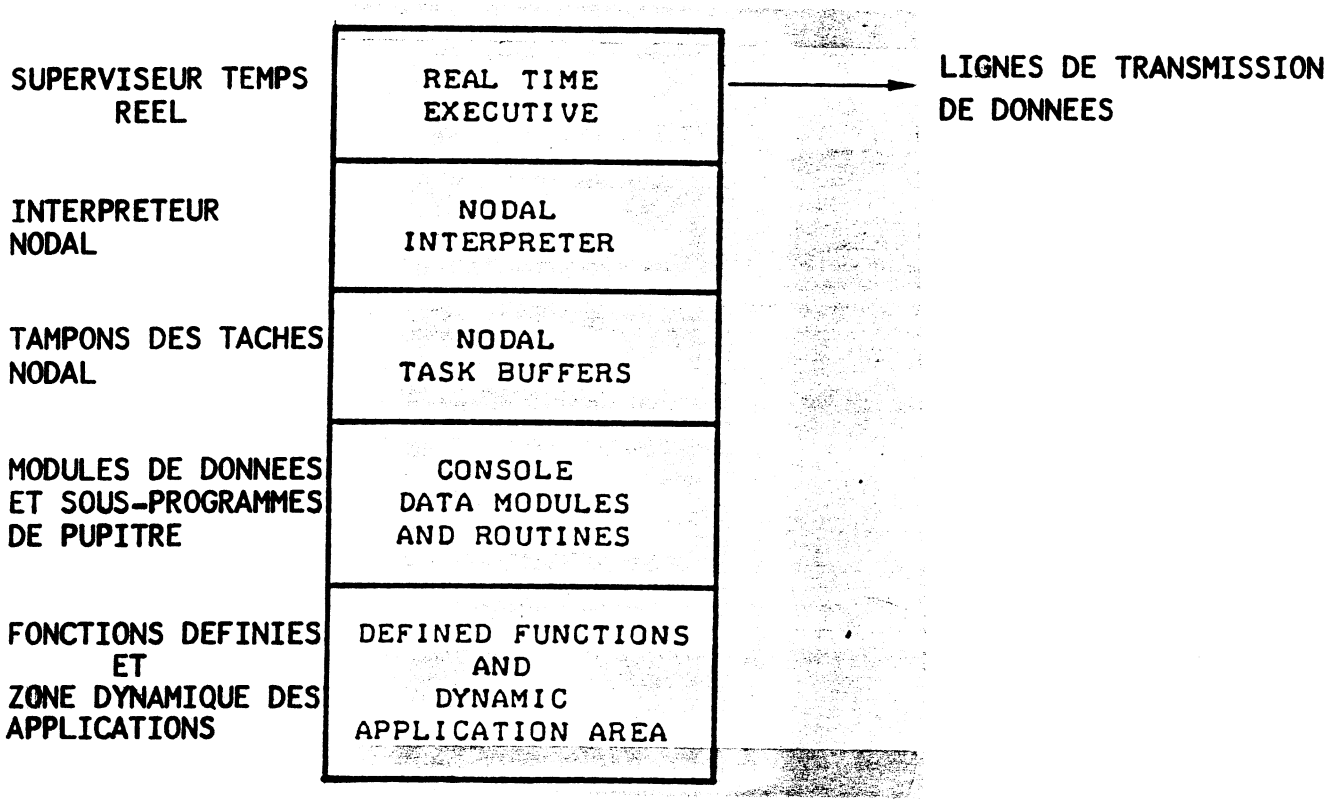
d) MODULES DE DONNEES ET SOUS-PROGRAMMES EN CODE MACHINE

Cette zone comprend des modules de données spéciaux programmés en langage d'assemblage et des sous-programmes en code machine permettant de

gérer l'appareillage relié à cet ordinateur particulier. On y accède par l'intermédiaire d'une liste d'éléments NODAL, en fonction de ce qui est requis par le programme NODAL actuellement en cours d'interprétation.

12.2 Système pour un ordinateur de pupitre

L'organisation de la mémoire pour un ordinateur de pupitre est présentée ci-dessous.



Cette organisation diffère du système simple décrit ci-dessus par l'addition de la ZONE DYNAMIQUE DES APPLICATIONS. Dans cette zone sont stockées des fonctions définies en NODAL, soit créées en utilisant l'ordre DEFINE, soit chargées à partir d'un fichier. Un fichier système spécial contiendra un grand nombre de fonctions définies du système. Lorsque l'interpréteur NODAL résidant dans l'ordinateur de pupitre rencontre un nom du type variable du système qu'il ne peut trouver dans ses propres listes, il explore ce fichier spécial pour y rechercher une fonction définie, identifiée par ce nom. Si elle est trouvée, elle sera chargée puis exécutée. La fonction restera ensuite dans la zone dynamique en vue d'une exécution ultérieure plus rapide.

Sur l'ordinateur de pupitre on dispose d'un ordre particulier pour le chargement dans la zone dynamique. Il s'agit de l'ordre USE, par exemple :

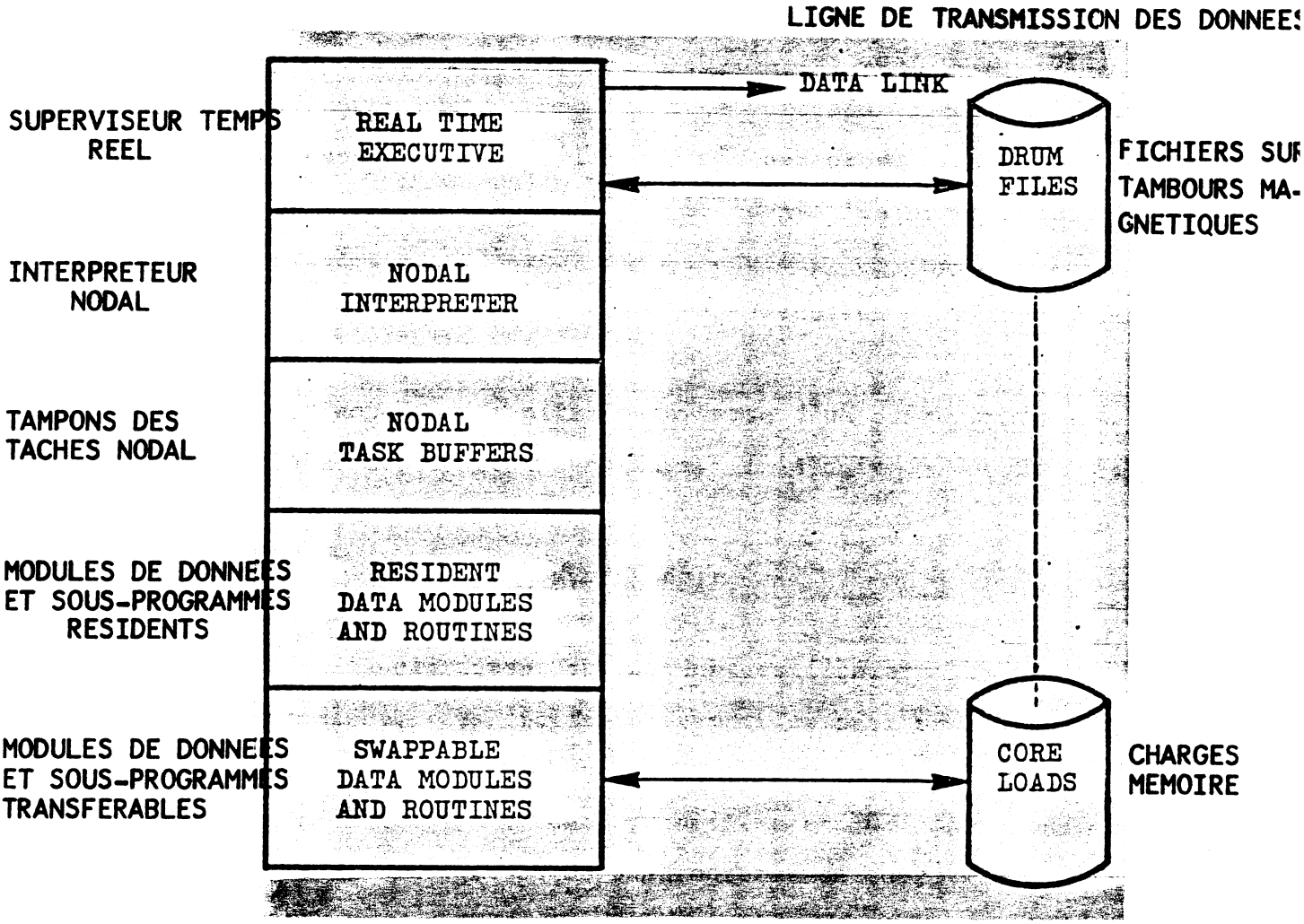
USE MYFILE RF1

entraînera le chargement du contenu des fichiers MYFILE et RF1 dans la zone dynamique. Ces fichiers peuvent contenir une sélection de fonctions NODAL pré-définies ou des sous-programmes d'application spéciaux rédigés en langage d'assemblage ou même en FORTRAN. Dans ce dernier cas, l'ordre USE agit comme un programme de chargement.

Il faut noter que cette configuration dynamique de l'ordinateur de pupitre peut correspondre à un processus lent. Il est prévu que le jeu de sous-programmes sera établi pour une application déterminée, par exemple au moment du début de la séance de calcul, puisqu'il restera plus ou moins constant. La zone dynamique peut être remise à zéro en utilisant l'ordre ZDEF.

12.3 Système pour un ordinateur polyvalent

Les systèmes utilisant des tambours magnétiques sont plus compliqués. Ils sont représentés schématiquement ci-dessous.



L'emploi d'un tambour magnétique offre deux possibilités supplémentaires. En premier lieu, il permet de disposer de fichiers locaux pouvant être stockés et contenant des programmes NODAL et des données; de ce fait le trafic sur la ligne de transmission de données se trouve réduit, ce qui est susceptible d'entraîner une réduction du temps de réponse par rapport à l'emploi de l'ordinateur gérant la bibliothèque des programmes; cela permet également une exploitation autonome. En second lieu un

tambour constitue une ressource du type "mémoire virtuelle". Cela permet de charger sur le tambour des sous-programmes en code machine ainsi que des tâches, qui y sont stockés sous une forme "image-mémoire", en étant prêts à être transférés chaque fois que cela est requis. Pour un tel transfert il faut disposer d'une zone supplémentaire en mémoire centrale, appelée zone de charge-mémoire.

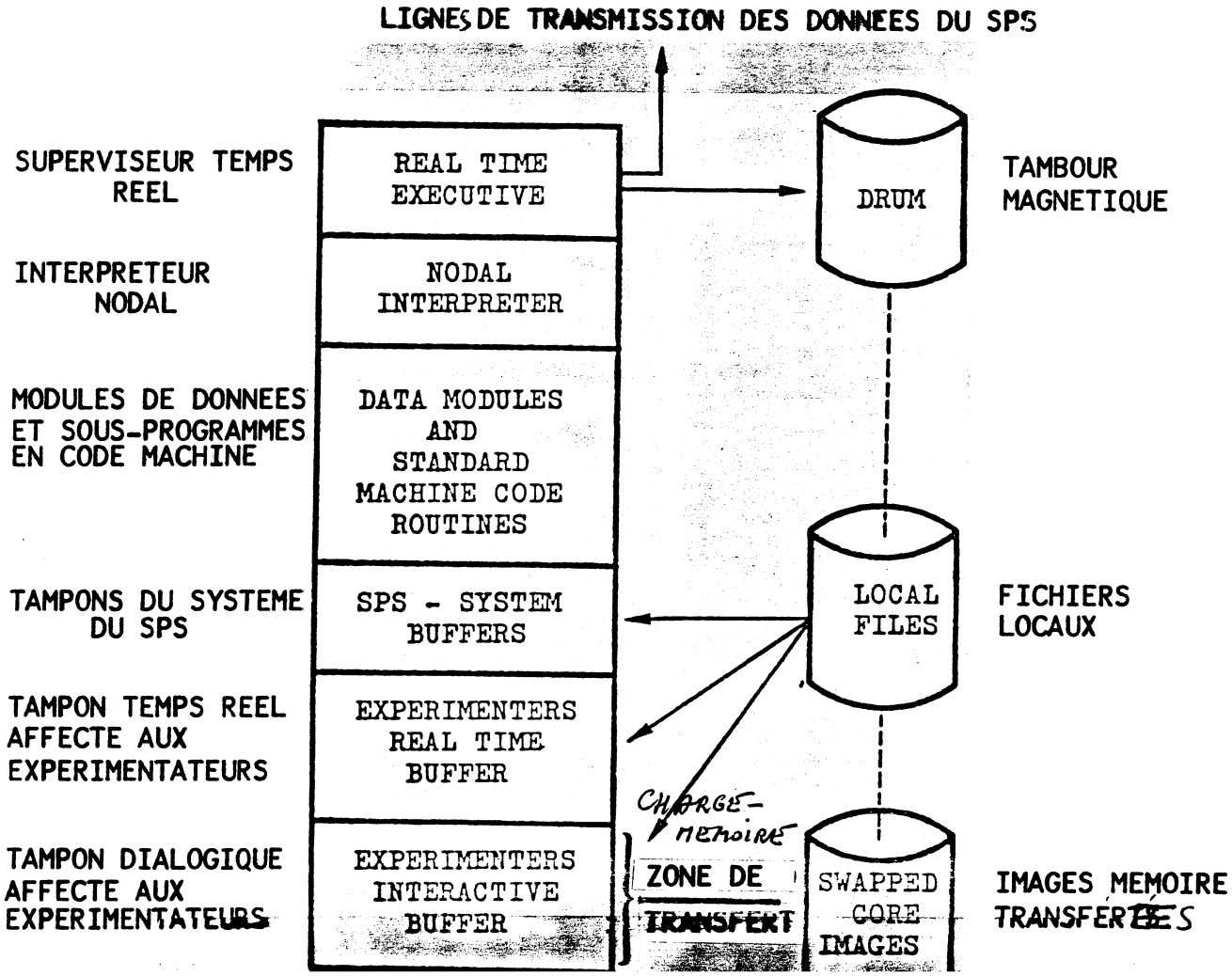
L'utilisation des charges-mémoires augmente le volume de code machine susceptible d'être employé dans un système. Un système typique peut disposer d'une zone de charge-mémoire de 4K mots et de huit charges mémoires de 4K. Cela permet d'utiliser 32K de programmes ou sous-programmes en code machine; bien qu'à un instant quelconque seuls 4K puissent être résidents. La pénalité est représentée par le temps de transfert. Il comprend un temps d'attente pour que le tambour magnétique atteigne sa position correcte, ainsi qu'un temps de transfert. Le tambour effectue une rotation complète en 20 ms, avec un débit de transfert de 2K par révolution. Ainsi la lecture d'une charge-mémoire de 4K exigera en moyenne 50 ms. Sur un disque, la lecture d'un fichier de 200 mots exigerait en moyenne 12 ms. Les charges-mémoires sont divisés en deux catégories: charges à lecture seule et charges avec écriture en retour. Si un programme stocke des données dans sa zone ou ailleurs sur la charge-mémoire, il faut que l'on puisse écrire en retour sur celle-ci. Pour une charge-mémoire de 4K, le temps de transfert se trouve ainsi augmenté et passe en moyenne à 100 ms.

12.4 Système pour un ordinateur affecté à une zone d'expérimentation

Le logiciel pour un ordinateur affecté à une zone d'expérimentation (EA) est différent du logiciel habituel du fait qu'il doit prendre en charge plusieurs utilisateurs travaillant en mode dialogique (par exemple, jusqu'à 10 utilisateurs). Le premier d'entre eux est la position de

commande de l'EA, bénéficiant probablement de la priorité la plus élevée. Ensuite chaque équipe d'expérimentateurs aura à sa disposition un terminal à partir duquel elle pourra faire exécuter des programmes en vue d'interroger l'état de la machine et de modifier en conséquence ses propres éléments de ligne de faisceau. De plus il sera fourni une liaison directe entre le système CAMAC associé à l'ordinateur d'une EA et le système CAMAC associé à l'ordinateur de l'expérimentateur. Cela permettra à ce dernier ordinateur soit de lire un bloc de données standard qui est mis à jour à la suite de chaque cycle, par l'ordinateur de l'EA, soit de lire un bloc de données spécial préparé séparément par un programme rédigé par l'expérimentateur, qui est exécuté sur l'ordinateur de l'EA.

Le schéma ci-après présente l'organisation du logiciel qui est proposée en vue de satisfaire les exigences précédentes.



Le MONITEUR TEMPS REEL gère l'accès aux fichiers stockés sur le tambour local et à la LIGNE DE TRANSMISSION DE DONNEES DU SPS; il contrôle le transfert des CHARGES-MEMOIRES, les priorités, etc. L'INTERPRETEUR NODAL se charge de l'exécution des programmes NODAL, habituellement en lançant des appels vers les MODULES DE DONNEES et les sous-programmes STANDARD EN CODE MACHINE.

Les TAMPONS DES TACHES sont utilisés pour stocker des programmes NODAL qui sont en cours d'interprétation, ainsi que la pile affectée, utilisés par l'interpréteur. L'interpréteur NODAL est rentrant, de sorte que plusieurs tampons des tâches peuvent être utilisés concuremment. Ici les TAMPONS DES TACHES sont divisés en deux lots. Le premier est réservé à l'usage du SPS. Dans ces tampons des ordres immédiats sont interprétés et des programmes provenant d'autres ordinateurs sont transmis au système du SPS; des programmes en boucle fermée et des programmes de surveillance y font également l'objet d'un ordonnancement de routine. L'autre lot de tampons des taches comprend le TAMPON TEMPS REEL AFFECTE AUX EXPERIMENTATEURS et les TAMPONS DIALOGIQUES. L'emploi de ces tampons est décrit en détail ci-dessous. Les TAMPONS DIALOGIQUES occupent tous les mêmes positions en mémoire centrale, mais ils disposent d'un emplacement unique sur le tambour magnétique. Selon les nécessités, ils sont transférés vers l'intérieur et vers l'extérieur par le système; le temps de transfert pour un tampon de 2K est d'environ 60 milliseconde

EMPLOI DIALOGIQUE

A partir de son terminal, tout utilisateur peut obtenir des informations sur sa ligne de faisceau, en utilisant les ressources standard de NODAL et les MODULES DE DONNEES contenus dans l'ordinateur de l'EA. De manière typique, il fera exécuter un programme qui doit afficher un tableau d'intensité de courant, de positions, de valeurs des échelles

de comptage, etc. Il peut également commander les éléments de sa ligne de faisceau, soit directement en utilisant des ordres immédiats, soit par l'intermédiaire de programmes NODAL effectuant des explorations, etc. La commande de l'équipement d'une ligne de faisceau s'effectue par l'intermédiaire des MODULES DE DONNEES et elle est validée par un système à mot de passe de sorte que chaque terminal ne dispose que d'un accès limité.

EMPLOI DE PLUSIEURS ORDINATEURS

Le système CAMAC d'un expérimentateur sera relié au système CAMAC de l'EA par un module de liaison. En écrivant un mot dans ce module, l'expérimentateur peut provoquer une interruption dans l'ordinateur de l'EA et obtenir une certaine réponse. Cette réponse peut prendre deux formes, selon le mot qui a été décrit. En premier lieu, un bloc standard de données de l'EA peut être renvoyé par l'intermédiaire du module de liaison. En second lieu, un programme spécifique NODAL, rédigé par l'utilisateur, peut être activé. Dans ce dernier cas l'ordre NODAL:

TRANSF(A)

renverra le contenu du tableau A (tableau de nombres entiers ayant une longueur maximale de 50 mots, par exemple) sur la ligne de transmission. Le programme NODAL qui effectue cette opération peut être rédigé selon les modalités normales et stocké dans un fichier du tambour local. Il est ensuite relié à l'interruption provenant de l'ordinateur de l'expérimentateur, en employant l'appel NODAL, par exemple:

HOOK(FILE,3.2)

où FILE représente le fichier dans lequel le programme a été stocké, alors que 3 est le numéro de l'utilisateur et 2 le numéro logique de

sa demande, déterminé en fonction du nombre qu'il a écrit en premier lieu dans le module de liaison afin de créer l'interruption. Voici certains ordres utilitaires:

DISCON(FILE)

déconnecte le chaînage (HOOK) établi ci-dessus, et:

ZTRANS

remet à zéro avec retour à l'état initial de la ligne de transmission à l'extrémité EA.

Résumé des ordres NODAL

Les ordres NODAL peuvent être abrégés, à la condition de n'introduire aucune ambiguité. Les ordres, abrégés ou non, doivent être suivis d'un espace (blanc) ou d'un certain autre caractère non numérique. Les trois premières lettres d'un ordre doivent toujours la définir sans ambiguité, mais parfois il est requis un plus petit nombre de caractères.

Dans le résumé qui suit, les lettres A, B, et C représentent des noms de variables, alors que X, Y et Z correspondent à une expression NODAL légale, quelconque, et que Ln est un numéro de ligne NODAL.

Ordre	Exemple de syntaxe	Explication
ASK	ASK A B C	NODAL affiche deux points pour chacune des valeurs requises; l'utilisateur frappe une valeur pour définir chaque variable
CALL	CALL SUB(X,A)	Transfère le contrôle au sous-programme défini par les paramètres X (valeur)ou A (référence)
DIMENSION	DIM A(3,2)	Dimensionnement d'un tableau de nombres réels
	DIM-I B(10)	Dimensionnement d'un tableau de nombres entiers
	DIM-S C(6)	Dimensionnement d'un tableau de chaînes de caractères
DEFINE	DEF-FUN INJ:VB DEF-SUB NAME(R-X) DEF-STR FCHR(S-X)	Définit un texte NODAL constituant un sous-programme, une fonction ou une fonction de chaîne de caractères
DO	DO 4.1	DO (exécuter) la ligne 4.1; après exécution retour à l'ordre suivant immédiatement l'ordre DO
	DO 4	Exécuter les ordres contenus dans toutes les lignes du groupe 4 jusqu'à ce qu'un RETURN soit rencontré. Retourner à l'ordre suivant immédiatement l'ordre DO

	DO X	Exécuter l'ordre DO comme cela est spécifié par la valeur de l'expression X
EDIT	EDIT 4.1	Préparation en vue de l'emploi des ordres d'édition de lignes
	EDIT 4.1,L	Si le "L" est présent, afficher la ligne spécifiée avant de procéder à l'édition
END	END	Termine l'exécution d'un programme
ERASE	ERASE 1.1 2 A B	Supprime des éléments dans le programme stocké dans la mémoire locale
	ERASE	Supprime toutes les variables
	ERASE ALL	Supprime tout ce qui a été introduit par l'utilisateur
EXECUTE	EX (X) 2 A	Transmet le groupe 2 du programme de l'utilisateur ainsi que l'élément A de la liste des variables du programme en vue d'une exécution sur l'ordinateur X
FOR	FOR A = X,Y,Z;(C) FOR A = X,Z;(C)	(C) est une suite d'ordres quelconques s'étendant jusqu'à l'extrémité de la ligne (et pouvant contenir d'autres ordres FOR); (C) est exécuté avec des valeurs de A comprises entre X et Z, incrémentées de Y. Si Y ne figure pas dans l'ordre, il est supposé égal à 1
GOTO	GOTO 3.4 GOTO X	Transfère le contrôle à la ligne 3.4 ou à la ligne repérée par l'expression X, arrondie à deux chiffres après le point décimal
IF	IF (X)Ln,Ln,Ln	Transfère le contrôle aux premier, second ou troisième numéros de ligne selon que X est négatif, nul ou positif

	IF(X) Ln,Ln; IF(X) Ln;	Transfère le contrôle selon les conditions actuellement satisfaites; sinon continue en exécutant l'ordre immédiatement ultérieur
	IF X>Y; --- IF Z<=Y; --	IF logique. Exécute le reste de la ligne si la condition est satisfaite Les conditions à satisfaire sont les suivantes: = < > <= >= <>
IMEX	IM(5) TYPE BCT(3)	Transmet un ordre à l'ordinateur 5 pour exécution immédiate. Résultat affiché sur le terminal qui a lancé l'ordre IMEX
LIST	LIST LIST 1.1 LIST 1 LIST 1.1 2 3	Frappe la totalité du programme NODA Frappe la ligne 1.1 Frappe toutes les lignes du groupe 1 Frappe la ligne 1.1 et les lignes des groupes 2 et 3
LOAD	LOAD FILE	Charge le contenu du fichier FILE dans le tampon des tâches. FILE peut contenir des données ou des lignes Tout élément correspondant dans le fichier des tâches sera supprimé et remplacé par l'élément homologue du fichier FILE
OLD	OLD FILE	Charge le programme contenu dans le fichier "FILE"
OPEN	OPEN FUNCTN	Copie le texte d'une fonction défini dans le tampon des tâches et supprim la fonction résidente.
OVERLAY	OVE FILE X,Y	Exécute le programme stocké dans le fichier FILE en tant que recouvremen (OVERLAY) dans le fichier des tâches Ensuite le programme disparaît. ARG (1) et ARG(2) prennent les valeurs X et Y

QUIT	QUIT	Renvoie le contrôle au moniteur
REMIT	REM A B	Renvoie les éléments de données A et B au programme qui a lancé le REMIT par l'intermédiaire d'un ordre EXECUTE
RETURN	RETURN	Retour depuis un groupe exécuté par un ordre DO
RUN	RUN	Exécute le programme qui vient d'être frappé
	RUN FILENAME	Exécute le programme stocké dans le fichier FILENAME
	RUN X FILENAME	Exécute à partir de la ligne X
SAVE	SAVE FILE	Mémorise le programme dans le fichier "FILE"
	SAVE FILE 2 A B	Mémorise les éléments spécifiés dans ce même fichier
SET	SET A = X	Affecte à la variable A la valeur de l'expression X
TYPE	TYPE A+B-C	Evalue l'expression et affiche le résultat
	TYPE [LIST]	Frappe la suite [LIST] qui peut contenir des expressions, des chaînes de caractères, des caractères de contrôle tels que: 1 pour retour de chariot avec avance d'une ligne # pour retour de chariot %X pour modifier le format et adopter le format X &X pour laisser X espaces (blancs)
USE	USE FILE	Ajoute le contenu de "FILE" à la liste des variables du système de l'ordinateur de pupitre

VALUE	VAL X	Sort d'une fonction NODAL avec X comme valeur
WAIT	WAIT (X)	Attendre des données qui seront transmises par l'ordinateur X
	WAIT-TIME X	Attendre X secondes
	WAIT-CYCLE 4	Attendre l'événement 4 du cycle de l'accélérateur
WHILE	WHILE A<2; DO 3	Exécute le reste de la ligne après le signe ";" si la condition WHILE est satisfaite
%	% THIS IS A COMMENT	Le reste de la ligne est considéré comme un commentaire
?ON ?OFF		Connecte (ON) ou déconnecte (OFF) la ressource pour l'affichage des étapes intermédiaires du traitement
$SET	$SET A="QWER"	Affecte le contenu QWER à la variable de chaînes de caractères A. Créé A lorsque cette variable n'existe pas déjà
	$SET A=INPUT(X):X	Affecte à la variable A le résultat de la fonction INPUT. En cas d'échec concernant cette fonction, va à la ligne représentée par X
$ASK	$ASK"STRING IS"A	Lit une chaîne sur le terminal (terminée par un retour de chariot); stocke cette chaîne dans A; créé A si cette variable n'existait pas déjà
$IF	$IF(A-B)Ln,Ln,Ln	Compare lexicalement les chaînes de caractères A et B. Effectue un branchement comme le IF arithmétique
$DO	$DO STRI	Exécute l'ordre stocké dans STR1
$VALUE	$VALUE "XYZ"	Crée la configuration P1, ici la chaîne de caractères "XYZ"

$MATCH	$M STRI P1 : X	Recherche une correspondance entre la chaîne STR1 et la configuration P1. S'il n'y a pas correspondance, va à la ligne repérée par X.

ORDRES POUR L'EDITION DE LIGNES

(CTRL)A	Retour arrière d'un caractère (frappe ↑)
(CTRL)C	Copie un caractère de l'ancienne ligne dans la nouvelle ligne
(CTRL)D	Copie le reste de l'ancienne ligne et va à la ligne
(CTRL)E	Modifie le mode insertion/remplacement (affiche des flèches ouvertes)
(CTRL)F	Copie le reste de l'ancienne ligne sans l'afficher et va à la ligne
(CTRL)H	Copie le reste de l'ancienne ligne, sans aller à la ligne
(CTRL)I	Crée un espace jusqu'au prochain arrêt défini par la tabulation
(CTRL)L	Va à la ligne
(CTRL)M	Va à la ligne, avec retour de chariot
(CTRL)OC	Copie l'ancienne ligne jusqu'au caractère C exclus
(CTRL)PC	Saute des caractères dans l'ancienne ligne jusqu'au caractère C
(CTRL)Q	Retour en arrière jusqu'au début de la nouvelle et de l'ancienne lignes
(CTRL)R	Réimprime rapidement
(CTRL)S	Saute un caractère dans l'ancienne ligne (affiche %)
(CTRL)T	Réimprime avec alignement
(CTRL)U	Copie jusqu'au prochain arrêt défini par la tabulation
(CTRL)VC	Prend le caractère C de manière littérale
(CTRL)W	Retour en arrière d'un mot (affiche \ )
(CTRL)XC	Saute des caractères dans l'ancienne ligne jusqu'à C inclus
(CTRL)Y	Ajoute le reste de l'ancienne ligne à la nouvelle ligne et édite le résultat
(CTRL)ZC	Copie l'ancienne ligne jusqu'au caractère C inclus.

LAB II-CO/Comp. Note 75-4
Juin 1975

PROGRAMMATION DES GENERATEURS DE CARACTERES "CHRISTIAN ROVSING"

1. INTRODUCTION

Un sous-programme particulier de commande de périphérique a été rédigé pour ces générateurs de caractères. Il traite en une séquence de 8 bits des caractères fournis par l'ordre NODAL: TYPE. En NODAL, la suite en sortie obtenue par exécution de cet ordre aboutit normalement au périphérique désigné par ODEV = 1, ce qui correspond en fait au sous-programme de commande VISTAR de l'écran utilisé en ligne. Cette suite en sortie peut être dirigée vers d'autres sous-programmes de commande de phériphériques, en utilisant la fonction ODEV. Par exemple, l'ordre:

```
1.1 SET ODEV=10
```

dirigera le flux en sortie vers le grand écran de visualisation, par l'intermédiaire du sous-programme de commande pour le module "Christian Rovsing". Des ordres ultérieurs, par exemple:

```
1.2 TYPE "ABCD"
```

entraîneront un affichage sur cet écran. Le périphérique de sortie peut être modifié en un point quelconque du programme; à la fin, il y a retour automatique à l'écran en ligne. Un ordre immédiat agit comme un programme en ligne; par exemple, les deux ordres immédiats suivants:

```
SET ODEV=10; TYPE "ABCD"
TYPE "EFGH"
```

entraîneront l'affichage de ABCD sur l'écran en couleurs principales, puis de EFGH, sur l'écran en ligne. Le jeu des caractères visibles

est présenté à l'Annexe 1. L'écran peut contenir 24 lignes de chacune 64 caractères. Il est prévu une écriture automatique d'une nouvelle ligne en haut de l'écran, tant à la fin d'une ligne que lorsque l'on atteint la dernière ligne. Les codes du sous-programme de commande, présentés à l'Annexe 2, s'appliquent également au car du sous-programme de commande du VISTAR en ligne. Il faut toutefois noter que le VISTAR a 80 caractères par ligne, qu'à la fin de la page il y a défilement du texte sur l'écran au lieu d'une écriture de la nouvelle ligne en haut d'écran, et que tous les codes des sous-programmes de commande n'y sont pas incorporés (en particulier, ceux qui ont trait à l'adressage absolu de la position et aux caractères spéciaux).

2. CARACTERES SPECIAUX

Le jeu des caractères visibles est présenté à l'Annexe 1. En plus des caractères usuels, il a été prévu certains caractères spéciaux pour des représentations graphiques à grande échelle. Ils permettent de tracer des schémas synoptiques. Par exemple, la suite d'ordres:

```
1.2 FOR I=1,10; TYPE \135
```

permet de tracer une barre horizontale de 10 caractères de long, à partir de l'actuelle position d'écriture. Il en est ainsi car 135 est le numéro de code d'une barre horizontale, comme cela est indiqué à l'Annexe 2. \135 est un exemple particulier des caractères de contrôle \X décrits dans le manuel NODAL, dans le cas de l'ordre TYPE, Cette suite indique à l'ordre TYPE de sortir le code de caractères correspondant à la valeur de l'expression faisant suite à la barre oblique \ (c'est-à-dire l'expression X, ou plus précisément 135, dans le cas actuel).

L'emploi de ces caractères spéciaux pour le tracé de schémas synoptiques, habituellement légendés en utilisant des caractères ordinaires, constitue un outil très puissant, incorporé dans le système du pupitre. Le jeu de caractères visibles se trouve complété par le mode de présentation "INVERTED". Au lieu d'un affichage en

lettres claires sur fond gris foncé, cet ordre fait apparaître des caractères de teinte foncée sur un écran à fond clair. Cela peut être utile dans le cas d'étiquettes ou de messages spéciaux.

3. CARACTERES DE CONTROLE

Le sous-programme de commande interprète certains codes de caractères comme représentant des caractères de contrôle. Par exemple, dans la suite:

```
1.2-TYPE \0 "RED"
```

le premier caractère à sortir a un numéro de code zéro. Cela indique au sous-programme de commande que tous les caractères ultérieurs seront à afficher en rouge ("RED"), et cela jusqu'à ce que lui parvienne une autre suite de contrôle entraînant une modification de la couleur.

Certains suites de contrôle exigent que le code de caractères ultérieur y figure sous forme de données; par exemple:

```
TYPE \11 \24
```

sont les deux codes de caractères ayant les numéros 11 et 24. Le code de caractères 11 indique au sous-programme de commande de modifier la position de la ligne d'écriture qui doit être celle qui est indiquée par la seconde suite de contrôle, soit ici: 24, correspondant à la ligne du bas.

Afin de faciliter l'emploi de ces fonctions de commande, certaines notations mnémoniques ont été introduites. Elles prennent la forme de variables de chaîne ou de fonctions. Il s'agit des ordres suivants:

ERASE efface le contenu de l'écran, établit la position d'écriture tout en haut, à gauche ,
définit la couleur: blanc sur fond non inversé (gris foncé)

RED	entraîne l'affichage des caractères ultérieurs en rouge
GREEN	entraîne l'affichage des caractères ultérieurs en vert
BLUE	entraîne l'affichage des caractères ultérieurs en bleu
WHITE	entraîne l'affichage des caractères ultérieurs en blanc
INVERT	entraîne l'affichage des caractères de teinte foncée sur un écran en fond clair, ou vice versa, selon le mode d'affichage actuel
LINE (Y)	déplace la position d'écriture jusqu'à la ligne Y
COLUMN(X)	déplacement sur la même ligne, jusqu'à la colonne X

4. EXEMPLE

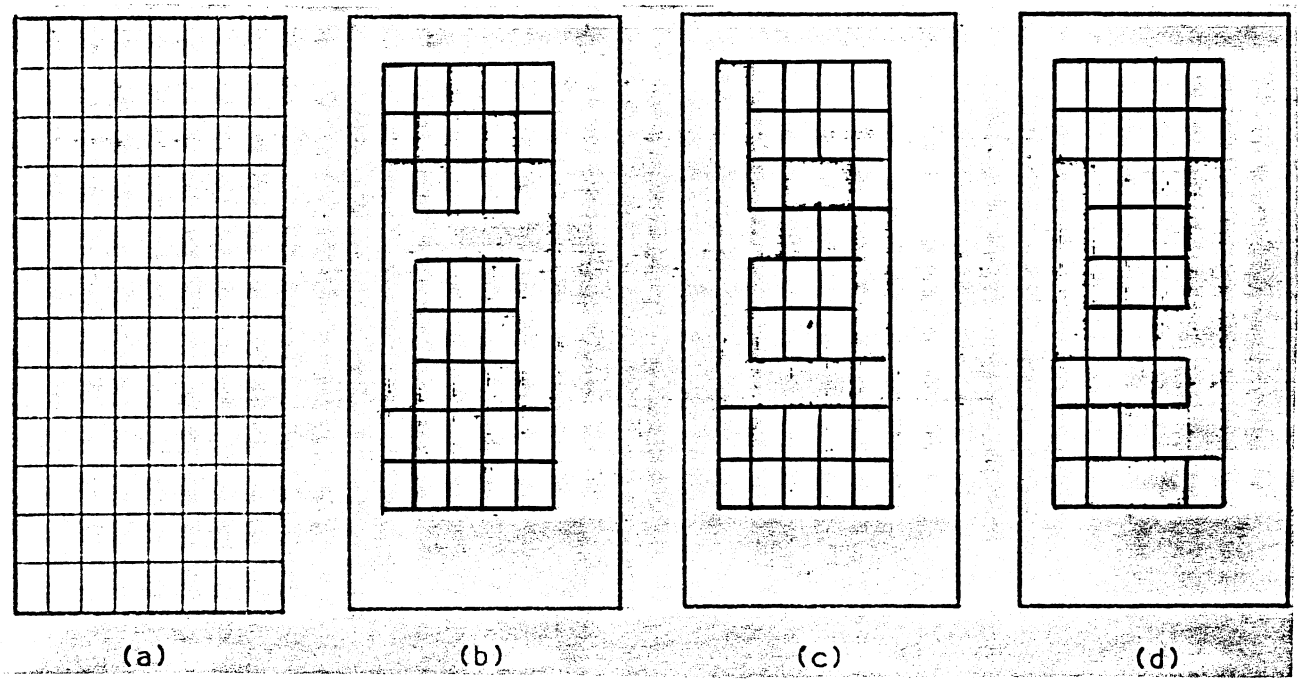
```
1.1  SET ODEV=COLOUR; TYPE ERASE
1.2  TYPE COLUMN (30)  "HEADING"!!
1.3  TYPE GREEN "MAGNET NO." COLUMN (40) WHITE "VALUE"!
1.4  FOR I = 1,64  TYPE \135
1.5  T\9\24\11\10  "COLUMN# 24, LINE 10"!
1.6  END
```

ANNEXE I

Jeu de caractères visibles "Christian Rovsing"

Les caractères visibles sont tracés dans un rectangle dont le rapport de la hauteur à la largeur est approximativement de 2:1. Le rectangle est divisé dans le sens vertical en 12 lignes pour affichage type TV, alors que dans le sens horizontal il y a 8 positions de points, comme le montre la figure (a) ci-dessus. Les caractères apparaissent sous forme de points dans cette matrice 8 x 12. Ainsi, la distance verticale entre les points est plus grande que leur espacement horizontal.

Les caractères normaux sont affichés dans une matrice réduite de 5 x 9 points, située à l'intérieur de la matrice 8 x 12, comme le montrent les figures (b), (c) et (d), ci-dessous:



Comme le montre la figure (b) ci-dessus, les caractères en majuscule n'occupent que les 7 lignes du haut de la matrice à 9 lignes. Les deux positions verticales restantes sont utilisées pour le tracé des parties inférieures des caractères en minuscules (p et g , par exemple), comme il est montré sur la figure (d) ci-dessus. Le jeu de caractères normal comprend les éléments suivants:

NOMBRES	0 à 9
LETTRES	MAJUSCULES: A à Z, minusucles: a à z
CARACTERES ASCII	! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~

CARACTERES SPECIAUX

Un certain nombre de caractères spéciaux ont été prévus. Ils permettent de tracer des schémas synoptiques simples, ne comprenant que des lignes horizontales et verticales. Ces caractères utilisant toutes les positions de points; ils sont présentés et désignés par leur numéro de code à la page suivante. Il faut noter que tous les caractères peuvent être inversés, c'est-à-dire qu'ils peuvent apparaître sous la forme de points de teinte foncée sur un fond clair. Cela est très utile pour les étiquettes et autres éléments de présentation. Cela signifie également que certains caractères spéciaux, par exemple un demi-carré inférieur (▄) ou une vanne horizontale, se transforment en d'autres caractères, soit ici, un demi carré supérieur (▀)et une vanne verticale (pour schémas de circuits de fluides).

ANNEXE 2

Codes de caractères pour un sous-programme de commande de générateur "Christian Rovsing"

N° DE CODE	CARACTERE AFFICHE OU EFFET
0	ROUGE à partir de maintenant
1	VERT à partir de maintenant
2	POURPRE à partir de maintenant
3	BLANC à partir de maintenant
8	Position extrême d'écriture, en haut et à gauche
9	Colonne de position donnée , chargée à partir du prochain caractère
10	Avance d'une ligne, (passage à la ligne suivante)
11	Ligne de position donnée, chargée à partir du prochain caractère
12	Effacement (ERASE) avec retour effectif en haut et à gauche (8) et affichage en blanc (3)
13	Retour du chariot (déplacement jusqu'à la colonne 1)
15	INVERSION (INVERT) (passage au mode fond inversé ou retour au fond normal, depuis ce mode)
20	Conserve position d'écriture et couleur actuelles
21	Restitue la position d'écriture et la couleur conservées
25	Incrémente la position de la colonne
26	Décrémente la position de la colonne
28	Décrémente la position de la ligne
29	Incrémente la position de la ligne
30	Ne pas transformer les caractères en minuscules
31	Transformer les caractères en minuscules
32	Espace (blanc)
33-47	! " # $ % & ' ( ) * + , - . /
48-57	0 - 9
58-64	: ; < = > ? @
65-90	A - Z
91-96	[\ ] ^ _ `
97-122	a - z
123-127	{ | } ~ !

128	Ligne verticlae
129	Ligne horizontale
130	Coin inférieur droit
131	Coin supérieur gauche
132	Coin supérieur droit
133	Coin inférieur gauche
134	Barre verticale
135	Barre horizontale
136	Té droit (⊣)
137	Té gauche (⊢)
138	Té inversé (⊥)
139	Té normal (⊤)
140	Croix
141	Demi-carré inférieur (⌊)
142	Vanne (fluidique)
143	Point central
144-157	Divers caractères particuliers

LAB II=CO/Comp. Note/75-5
Juin 1975

EMPLOI DE L'AFFICHAGE EN COULEURS SUR LE PUPITRE

1. INTRODUCTION

L'écran principal pour affichage en couleurs constitue la ressource de visualisation la plus puissante du pupitre; il doit donc être utilisé par les principaux programmes de commande. Trois types d'informations peuvent apparaître sur l'écran, savoir:

CARACTERES EN COULEURS
REPRESENTATIONS GRAPHIQUES
CURSEUR

Comme emploi typique de cet écran on peut citer le tracé d'un schéma synoptique en utilisant la ressource de présentation de graphique avec des légendes composées de caractères en couleurs, conformément à la convention adoptée pour le SPS (référence 1), et sélection d'éléments en utilisant le curseur. Après sélection, ces éléments peuvent être contrôlés à l'aide du bouton, avec affichage immédiat des valeurs correspondantes sur l'écran en couleurs. On peut également utiliser les touches à effleurement afin d'obtenir d'autres options de traitement concernant l'élément sélectionné.

2. COMPOSITION DES AFFICHAGES

Afin de faciliter la composition de telles présentations, il a été préparé une notice intitulée "DISPLAY LAYOUT SHEET" (feuille pour la composition des affichages). A la fin de cette note, on trouvera une version réduite de cette notice. La notice réelle est en format A3; on peut en obtenir des copies auprès du secrétariat du groupe "commandes".

La zone d'affichage est divisée en 24 lignes de 64 caractères. Les caractères peuvent apparaître en une position quelconque, sous le

contrôle du programme, comme cela est décrit dans la référence 2. Des caractères sur fond inversé (fond clair) sont très utiles pour la confection d'étiquettes légendant des schémas. Des schémas synoptiques simpl peuvent être tracés en utilisant une série de caractères spéciaux. Le jeu des caractères visibles est présenté à l'Annexe 1.

Des diagrammes et des graphes peuvent également être tracés par points dans la zone d'affichage en utilisant la ressource de présentation graphique. Le système de tracé est basé sur l'emploi d'une matrice de points de 720 x 576 points dont toutes les positions peuvent être adressées en tant que points compris entre (0,0) et (719, 575), comme cela est montré sur l'échelle des graphiques figurant dans la notice pour la composition des affichages. Des fonctions sont également disponibles pour le tracé par points de vecteurs, cercles, fonctions, histogrammes , axes, etc. Elles sont décrites dans la référence 3.

Le curseur peut apparaître en jaune sur l'écran et peut avoir diverses formes, à nouveau décrites dans la référence 3. Le curseur peut être automatiquement chaîné à la boule, et sa position peut être lue en la définissant sous forme de positions de caractères ou à l'aide de coordonnées sur un graphique. La fonction CFIND offre une ressource utile pour l'emploi du curseur. Elle permet de définir un certain nombre de positions discrètes sur l'écran, de telle sorte que CFIND donnera le numéro de l'élément dont le curseur se trouve actuellement le plus rapproché.

3. PROGRAMMATION DES AFFICHAGES

L'écriture de caractères en couleurs est commandée par l'ordinateur de pupitre. Un générateur de caractères en couleurs est connecté en permanence à cet ordinateur ainsi qu'à l'écran principal en couleurs. Par exemple, lors de son exécution dans l'ordinateur de pupitre, la suite

```
1.1 SET ODEV=COLOUR
1.2 TYPE ERASE LINE(12) COLUMN(30) "ABCD"
```

effacera le contenu de l'écran et écrira en blanc les caractères ABCD au centre de l'écran. Dans la référence 2 on présente des détails de la programmation du générateur de caractères.

L'ordinateur affecté à l'affichage permet d'obtenir des représentations graphiques. Par exemple, le programme :

```
1.1 EX (DISP) 2; END

2.1 BREAK(10); SET DDEV=11
2.2 MOVE(0,0); VECT(719,575)
```

tracera une ligne rouge en diagonale à travers l'écran. L'ordinateur d'affichage opère par tracé de points ou de vecteurs sur un "convertisseur d'image" produisant un signal vidéo TV qui peut alimenter un dispositif d'affichage comprenant un moniteur de TV. Actuellement l'écran couleurs principal reçoit des signaux fournis par trois dispositifs d'affichage donnant respectivement les couleurs rouge, vert et bleu. Les ordres d'affichage tels que ceux qui figurent à la ligne 2.2 ci-dessus s'appliquent au dispositif d'affichage actuellement employé, tel qu'il a été choisi à l'aide de la fonction DDEV. Dans l'exemple ci-dessus:

```
SET DDEV=11
```

les ordres contenus dans la ligne 2.2 concernent la couleur rouge. Les nombres 12 et 13 de DDEV peuvent être utilisés pour des affichages en vert ou en bleu.

La situation se trouve quelque peu compliquée par le fait que l'ordinateur de pupitre ne dispose que de quatre convertisseurs d'image alors que sur chaque pupitre il y a 8 écrans d'affichage! De ce fait, la fonctic DDEV commence par vérifier si un de ces convertisseurs est connecté à un dispositif d'affichage. Dans le cas contraire, elle en prend un dans la réserve et le connecte à ce dispositif. S'il n'y a aucun convertisseur d'image qui soit libre, un message d'erreur est transmis.

Le curseur est programmé et commandé par l'ordinateur de pupitre, en utilisant les fonctions décrites dans la référence 3. Toutefois la connection de ce même curseur est gérée par l'ordinateur de pupitre, en employant la fonction CURCON; par exemple:

```
CURCON(10)
```

connectera le curseur à l'écran couleurs principal étant donné que 10 est l'identificateur général de cet écran. CURCON (14) connecterait le curseur au grand écran noir et blanc, en le déconnectant de l'écran en couleurs, étant donné qu'il n'est pas souhaitable que ce curseur apparaisse simultanément sur deux écrans.

L'écran couleurs principal est donc ainsi un dispositif compliqué auquel peuvent être connectées plusieurs sources de signaux. En premier lieu, le générateur de caractères de pupitre ODEV=COLOUR=10, est connecté en permanence. De plus, des convertisseurs d'image peuvent être chaînés à différentes couleurs (actuellement, rouge, vert et bleu). Enfin, le curseur peut être connecté. Toutefois, chaque programme d'application peut exiger un ensemble de connections particulier. Pour ce faire, le procédé le meilleur consiste à utiliser la fonction BREAK dans l'ordinateur de console; elle déconnecte toutes les connexions établies avec un écran donné. Ainsi BREAK (10) déconnecte tout ce qui était relié à l'écran couleurs principal, étant donné que 10 est l'identificateur général de cet écran. De manière analogue, BREAK (14) déconnecterait tout ce qui était relié au grand écran en noir et blanc. La fonction BREAK doit être utilisée au début d'un programme, puis on doit établir les diverses connexions souhaitées. Lorsqu'un convertisseur d'image est déconnecté du dernier écran employé, il se trouve libéré et il est ramené dans la réserve de convertisseurs de l'ordinateur de pupitre.

4. REFERENCES BIBLIOGRAPHIQUES

1. Proposed colour convention for the main control consoles
(Code de couleurs proposé pour les principaux pupitres de commande)
E.J.N. Wilson, LAB II-DI-PA/EJNW/pd,
11 février 1975

2. Programming the Christian Rovsing character generators
(Programmation des générateurs de caractères "Christian Rovsing")
LAB II-CO/Comp. Note 75-4

3. NODAL user functions. Short descriptions and parameters format. Consol
Display system
(Fonctions NODAL à la disposition de l'utilisateur. Brèves descriptions et formats des paramètres. Système de pupitre pour l'affichage).
LAB-CO/Comp. Note 75-3

Lab.II-CO/Int./Comp.Note/75-13

L'ordinateur de service et son système de transmission de messages, SINTRAN III

O. Sveen

GUIDE DE L'UTILISATEUR

1. Comment utiliser les programmes dialogiques NODAL
2. Comment utiliser l'ordinateur de service à partir d'autres ordinateurs
3. Comment lancer l'exécution d'une tâche sur l'ordinateur de service (procédure de démarrage)

1. Comment utiliser les programmes dialogiques NODAL en temps réel

1.1 Procédure pour lancer l'exécution d'un programme NODAL

Afin d'être autorisé à lancer l'exécution d'un programme dialogique NODAL en temps réel, l'utilisateur doit débuter une séance en se faisant connaître comme utilisateur du système (user SYSTEM) ou utilisateur travaillant en temps réel (user RT). La procédure de début de séance est décrite en détail dans le "SINTRAN III USERS' GUIDE" (Guide des utilisateurs de SINTRAN III); pour l'essentiel, elle est identique à celle qui est employé avec le système pour l'exploitation en temps partagé (TSS). Pour être autorisé à l'entrée, il est nécessaire de connaître un mot de passe.

Les deux types d'utilisateurs, "RT" et "SYSTEM" ont la faculté d'émettre un certain nombre d'ordres qui sont illégaux pour des utilisateurs d'arrière-plan habituels. L'un de ces ordres est : RT< nom >; il lance le programme en temps réel désigné par < nom > . Dans l'ordinateur de service il existe autant de programmes dialogiques NODAL en temps réel qu'il y a de terminaux. Un de ces programmes est lancé en frappant au clavier RT NODxx, où xx est le numéro de terminal sur lequel le programme NODAL est requis. Le numéro du terminal est trouvé en utilisant l'ordre WHO-IS-ON. Lorsque l'ordre en temps réel, RT, a été exécuté, le programme NODAL qui a été lancé tente de réserver le terminal. Si ce terminal est réservé pour un processus d'arrière-plan, il est nécessaire de transmettre un ordre de fin de séance (comme pour le système d'exploitation en temps partagé). Le programme NODAL se trouve ainsi libéré de son état d'attente, et il apparaît sur le terminal avec son en-tête habituel. Le terminal sera maintenant réservé par NODAL jusqu'à l'achèvement de la séance ("Quit") ou l'abandon de l'exécution. Ensuite il est à nouveau possible de lancer un ordre de début de séance en tant qu'utilisateur d'arrière-plan.

1.2 Comment accéder aux fichiers à l'intérieur ou à l'extérieur de l'ordinateur de service

Tous les accès aux fighiers à partir de programmes en temps réel en SINTRAN III utiliseront les fichiers appartenant à l'utilisateur d'arrière-plan RT lorsqu'aucun autre nom d'utilisateur a été spécifié explicitement. Un nouveau fichier est spécifié en indiquant son nom placé entre les guillemets, comme pour l'exploitation en temps partagé. En indiquant un nom d'utilisateur (voir le manuel :"NORD FILE SYSTEM": Système de fichiers NORD), il est possible de lire des fichiers d'autres utilisateurs, mais non pas d'y écrire, à moins que l'utilisateur ait

fait du programme son "ami" ("friend"). Il est également possible de permettre aux programmes NODAL en temps réel de créer des fichiers appartenant à d'autres utilisateurs que RT, en utilisant l'ordre d'arrière-plan: SET-FRIEND-ACCESS. Toutefois, à partir de programmes NODAL en RT on ne peut pas effacer des fichiers déjà créés (voir : "NORD FILE SYSTEM"). Si un nom de fichier est précédé d'un nom d'ordinateur valide, placé entre des parenthèses (par exemple: (LIB)), le fichier correspondant est considéré comme étant dans la mémoire de masse de l'ordinateur spécifié. De cette manière on peut lire et écrire des fichiers NODAL résidant dans tous les volumes de la mémoire de masse dans le système de commande (Control System), à la condition que le système de protection le permette. Si l'on a indiqué un nom d'ordinateur non existant, NODAL tentera de trouver dans le système de fichiers de l'ordinateur de service un utilisateur ayant le même nom.

Les périphériques de l'ordinateur de service (L-P = imprimante par lignes F-P = perforatrice rapide; T-R = lecteur de bandes; M-T = bande magnétique; C-R = lecteur de cartes) peuvent être réservés par un programme NODAL en utilisant la fonction de chaîne : OPEN. L'ordre : SET ODEV = L-P sera insuffisant, mais : SET ODEV = OPEN ("W", "L-P") dirigera le flux de données en sortie vers l'imprimante par lignes.

1.3 Emploi de la ressource pour copie d'image-mémoire

Le programme NODAL RT de l'ordinateur de service comprend une fonction, BINCOP, qui copie des images-mémoire d'un fichier de l'ordinateur de service dans un fichier d'ordinateur de la bibliothèque. Le fichier source doit être un fichier de l'ordinateur de service décrit avec un ordre ")BPUN" en assembleur MACF. Le fichier destinataire peut être

un fichier quelconque du système d'ordinateurs, à la condition que sa capacité soit suffisante. Le fichier ")BPUN" est copié sans altération, à l'exception de l'amorceur en octal qui est supprimé. Le premier mot du fichier destinataire contiendra "space!", avec l'information binaire contenue dans les mots suivants. Le format de l'appel BINCOP est identique à celui d'un ordre COPY, soit :

BINCOP < nom du fichier destinataire > < nom du fichier source >

par exemple :

BINCOP (LIB) < 24 > ALARM (SYSGEN) SYSTEM : BIN

1.4 Ressources pour exécution de travaux à distance

Un programme NODAL en temps réel (RT-NODAL) de l'ordinateur de service peut transmettre des tâches pour exécution sur d'autres ordinateurs. Les ordres EXECUTE, EXEC-PRIOR et IME sont utilisés dans les mêmes conditions que lorsque l'on travaille sur les pupitres, à une exception près : les ordinateurs doivent être spécifiés par des noms et non par des numéros.

Dans le logiciel de la ligne de transmission de données de l'ordinateur de service, il est incorporé une ressource gérant l'expiration des délais. Si un programme NODAL, attendant que lui soient transmises des données, n'a rien reçu au bout de 8 secondes, il est dérivé vers un retour sur erreur (FILE ERROR = erreur de fichier). Le minuteur peut être arrêté à l'aide de l'ordre d'arrière-plan: "DSCNT INIDL", et lancé à l'aide de : "INTV INIDL 1 2" suivi de : "RT INIDL".

1.5 Eléments de protection dans les programmes NODAL en temps réel

Lorsque le système de protection aura été défini sous sa forme final et mis en service pour les autres ordinateurs du réseau, des fonctions analogues seront incorporées dans les programmes NODAL de l'ordinateur de service.

Actuellement seule une fonction de lecture/écriture, SECTN, est disponible. SECTN est utilisée pour établir les valeurs des mots de section et des mots descripteurs qui accompagnent les tâches devant être exécutées à distance sous le contrôle de ce programme NODAL. Le descripteur a toujours une valeur établie à 177777, par un ordre : SET SECTN = valeur.

2. Comment utiliser l'ordinateur de service à partir d'autres ordinateurs

L'ordinateur de service dispose d'une réserve de programmes NODAL en temps réel (RT-NODAL) dans laquelle on peut puiser à l'aide des ordres SCHEDL et HOOK, ainsi que pour l'exécution à distance des tâches. Lorsqu'une telle tâche entre dans l'ordinateur de service, un programme NODAL libre est trouvé dans cette réserve, et il est lancé en vue de l'exécution de la tâche. Des tâches EXECUTE lancées à distance ont des priorités encore plus élevées. Des tâches dialogiques d'arrière-plan SINTRAN III correspondant à de courts intervalles ont une priorité plus élevée que des programmes NODAL dialogiques ou EXEC; il est ainsi possible d'abandonner l'exécution de programmes NODAL susceptibles de bloquer le système. Toutefois il est impossible d'arrêter une tâche IMEX exécutant l'ordre : WHILE 0 = 0.

Tous les programmes NODAL en temps réel peuvent être supervisés

à l'aide des ordres d'arrière-plan LIST-EX-QUEUE et LIST-RT-DESCR. Pour obtenir des détails supplémentaires, on consultera le "SINTRAN III Users Guide".

Si la ligne de transmission de données jusqu'à l'ordinateur de service est indisponible, on doit essayer les ordres :

RT CLODL et
RT OPNDL, sur l'ordinateur de service ainsi que,
*CLODL 23 et
*OPNDL 23, sur l'ordinateur de gestion des messages.

Si aucun de ces ordres ne semble avoir d'effet, il convient de redémarrer le système SINTRAN III de l'ordinateur de service.

3. Comment lancer l'exécution d'une tâche sur l'ordinateur de service (procédure de démarrage)

a) Appuyer sur les touches "stop" et "master clear" du pupitre de l'ordinateur.
b) Ecrire "20500$" sur le terminal 1 de l'ordinateur de service. SINTRAN III est alors chargé à partir du disque et il est lancé. Il affiche : "SINTRAN III RUNNING" (SINTRAN III en activité).
c) Débuter la séance de travail dans les conditions habituelles pour tout terminal, à l'exception du terminal 1. Vous obtenez le message : "NO USER ENTERED" (aucun utilisateur entré).
d) Frapper au clavier : "E-D P-ONE DI F". Le disque fixe contenant le repertoire appelé "PACK-ONE" est inséré dans le système.
e) Indiquer au système l'heure et la date correcte en frappant : "UPDAT <minutes> <heure> <date> <mois> <année>".

f) Terminer la séance en frappant LOG.

g) Ouvrir la séance en tant qu'utilisateur SYSTEM (il vous faut connaître son mot de passe). Frapper : "MODE ENTER TER". Le fichier de mode, ENTER, effectuera les opérations suivantes :

1. Chargement du répertoire appelé PACK-TWO sur le disque mobile.
2. Affecte à PACK-TWO une valeur de "répertoire par défaut".
3. Emet l'ordre RTENTER faisant de RT un utilisateur de programmes en temps réel.
4. Etablit l'emplacement des programmes de la ligne de transmissio de données qui sont ainsi chargés sur des adresses contigues en mémoire, avec établissement d'une correspondance entre les adresses logiques et physiques de ces programmes.
5. Emet l'ordre "RT INIDL" qui initialise le logiciel de la ligne de transmission de données.

h) Les messages : "MHP SI III RUNNING"
"OPEN INPUT"
"OPEN OUTPUT"

apparaîtront sur le terminal 1. Si le dernier message n'apparaît pas automatiquement, vous devez frapper "*OPNDL 23" sur l'ordinateur de gestion des messages, afin de l'obtenir. A la suite de ces opérations, le système doit être prêt à lancer l'exécution de programmes NODAL en temps réel et à accepter des tâches pour exécution à distance.