SOMMAIRE (suite)

SOMMAIRE
(suite et fin)

1. Généralités

1-1. INTRODUCTION

MITRA 15 est un ordinateur temps réel qui par sa conception modulaire et sa structure microprogrammée très moderne peut aborder avec efficacité de nombreux domaines d'applications : automatisation de processus industriel, calcul scientifique, téléinformatique ou gestion de transactions.

Autour d'une mémoire vive à architecture plane s'organisent de 1 à 4 unités de traitement microprogrammées au moyen de mémoires mortes à lecture non destructive (ROM). Selon le type des microprogrammes intégrés aux unités de traitement ces dernières deviennent des unités centrales, des unités d'échange, ou des unités spécialement adaptées à des applications particulières. Chaque unité de traitement possède un MINIBUS permettant la connexion d'une gamme complète de périphériques.

MITRA 15 est présenté en deux modèles totalement compatibles qui ne se distinguent que par leur puissance de traitement et leur capacité de couplage à des équipements périphériques. Chaque utilisateur peut donc déterminer le modèle et la configuration qui conviennent le mieux à son application.

1-2. CARACTERISTIQUES ESSENTIELLES DE MITRA 15

1-2.1. Mémoire principale

La mémoire principale de MITRA 15 est une mémoire à tores de ferrites au lithium organisée en mots de 16 bits + 1 bit de parité et 1 bit de protection. Le cycle de base, très rapide, est de 800 nanosecondes par mot, ce qui représente un débit de 2,5 millions d'octets par seconde.

La mémoire est adressable par octet et altérable par octet, mot ou double-mot.

La mémoire est composée de blocs de 4 096 mots, c'est-à-dire 8 192 octets, le nombre maximum de blocs est de huit. Il est donc possible d'étendre la mémoire de 4 096 à 32 768 mots par bloc de 4 096 mots.

- Protection mémoire dynamique

La protection mémoire permet de protéger toute une zone de la mémoire contre les tentatives intempestives qui pourraient venir perturber la zone à protéger. Cette protection est modifiable dynamiquement (instruction LDP).

A chaque mot mémoire est associé un "verrou" de protection de 1 bit. Par ailleurs l'état programme comporte un indicateur dont la fonction est celle d'une "clef" : si cette clef est à 1, le programme peut accéder à toute la mémoire ; si elle est à zéro, le programme ne peut accéder qu'aux zones non protégées de la mémoire.

■ Parité

C'est un contrôle complet de parité aussi bien en mémoire que pour les Entrées/Sorties.

I-2.2. Unité de traitement

Les fonctions traditionnelles d'un ordinateur sont réparties entre un module câblé identique pour chaque unité de traitement et qui constitue en fait une micromachine, et le contenu de la mémoire de commande qui spécialise cette micromachine pour lui faire remplir les fonctions d'unité centrale, d'unité d'échange ou d'unité d'échange spécialisée.

Une unité de traitement se compose d'un bloc de registres rapides, de cinq indicateurs programme, d'une mémoire morte micro-programmée, d'un opérateur et d'un système d'interruptions et de suspensions.

■ Mémoire à registres rapides

Cette mémoire registre est réalisée en circuits intégrés bipolaires de type MSI, elle est organisée en blocs de 8 registres de 16 bits adressables par programme. Sa capacité est de 8 ou 16 blocs de 8 registres (soit 64 ou 128 registres) par unité de traitement. Temps d'accès : 60 nanosecondes pour 1 mot.

Pour l'unité centrale le premier bloc (bloc 0) est affecté au contexte programme, les autres blocs sont affectés aux échanges avec les périphériques.

Pour l'unité d'échange, tous les blocs sont affectés aux échanges avec les périphériques.

Bloc 0 de l'UC

P	Compteur programme
L	Base locale
G	Base générale
A	Accumulateur
E	Extension de A
X	Index

} Contexte programme

■ Indicateurs programme

C Report ou test opération
O Débordement ou test opération
MS Maitre/Esclave
MA Masque des interruptions
PR Protection mémoire

Les indicateurs programme sont étudiés plus en détail au chapitre II.

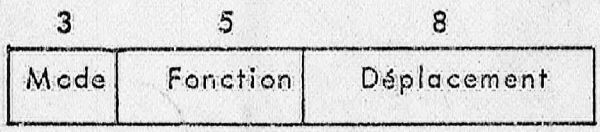
■ Instructions

MITRA 15 dispose d'un jeu de 87 instructions dont :

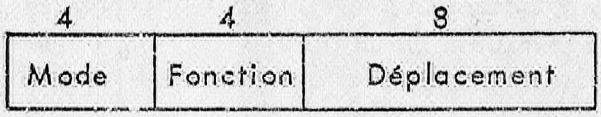
- 40 instructions à référence mémoire
- 29 instructions sur registre

- 12 instructions de décalage
- 6 instructions spéciales.

Toutes les instructions ont un format fixe :



ou



Elles portent sur octet, sur mot, sur double-mot et sur chaînes de caractères de longueur quelconque.

Les types d'adressages sont les suivants :

- adressage immédiat pour les opérandes qui peuvent être codés sur un octet;
- adressage direct, indirect et indexé par rapport à la base locale;
- adressage direct, indirect et indexé par rapport à la base générale.

■ Mémoire morte microprogrammée

C'est une mémoire à lecture non destructive (ROM : Read Only Memory) qui est préenregistrée. Chaque mot de 16 bits représente une micro-instruction. Elle est réalisée en circuits intégrés bipolaires de type MSI, le temps d'accès est de 60 nanosecondes. Sa capacité est de 512 ou 1 024 mots par unité de traitement.

Elle existe en trois versions avec les fonctions suivantes :

MC1 : exécute le code d'instructions de base et les fonctions de couplage des périphériques uniquement connectables sur le Minibus Unité Centrale.

MC2 : exécute le code complémentaire (instructions optionnelles) et les fonctions de couplage des périphériques connectables soit sur l'Unité Centrale, soit sur une Unité d'Echange.

MC3 : exécute les fonctions de couplage des périphériques uniquement connectables sur une Unité d'Echange.

■ Interruptions

MITRA 15 dispose de 32 niveaux d'interruptions prioritaires qui peuvent être armés, masqués, déclenchés par programme. Par regroupement, on peut disposer de 112 interruptions externes.

Le déclenchement d'une interruption provoque un changement automatique du contexte programme en 30 microsecondes.

Un niveau spécial rapide provoque un changement par commutation de bloc registres en 5 microsecondes.

■ Suspensions

MITRA 15 dispose également de 32 signaux de suspension hiérarchisés qui sont affectés au couplage par microprogramme des périphériques pour lesquels les transferts sont soit urgents, soit fréquents.

Leur temps de prise en compte maximum est de 300 nanosecondes.

■ Minibus

Chaque Unité de Traitement possède un minibus périphérique sur lequel viennent se connecter les coupleurs des périphériques. Ce minibus se présente sous la forme d'un circuit imprimé placé dans un fond de panier et permettant de banaliser entièrement l'implantation des cartes coupleurs.

■ Unité centrale MITRA 1523 comprenant :

- 1 024 mots de mémoire microprogrammée ROM
- 64 registres rapides
- Code étendu exécutant 87 instructions
- Système d'interruptions prioritaires
- MUL/DIV câblées
- Protection contre défauts secteur

■ Mémoire principale :

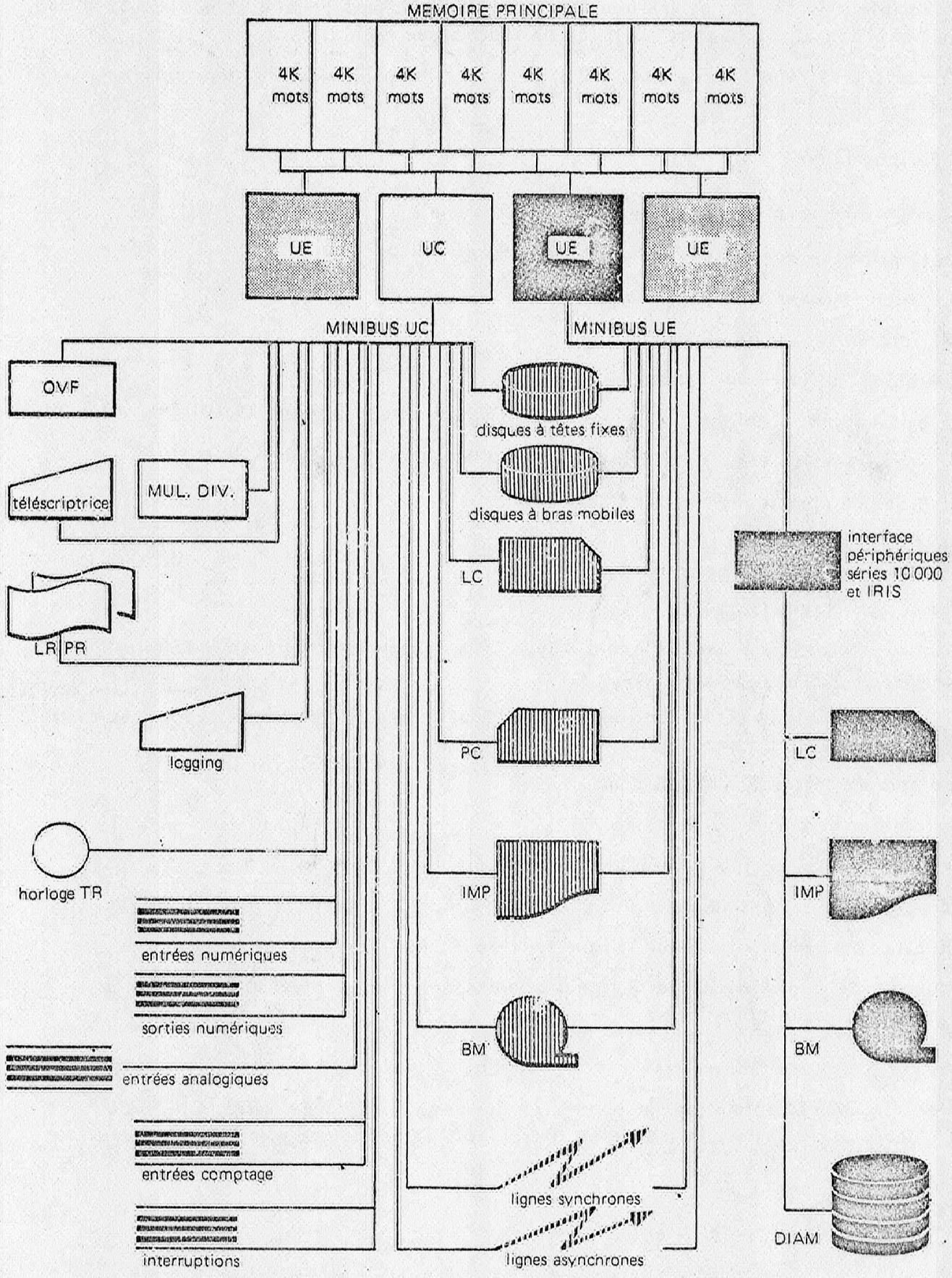
4 K à 32 K mots de 16 bits par module de 4 K mots

■ Performances :

- Modes d'adressage direct, indirect, indexé, relatif, immédiat, local, général.
- 1 index, 2 bases
- 87 instructions dont 40 à référence mémoire
- Chargement, lecture, écriture, addition mot en 2,1 µs
- MUL/DIV en 7 µs et 8 µs

■ Options principales :

- 1 à 3 accès direct à la mémoire
- 1 à 3 unités d'échange
- Extension à 128 registres rapides par unité de traitement (par module de 64)
- 32 niveaux d'interruptions prioritaires par modules de 1 ou 4 niveaux.
- Opérateur virgule flottante



Structure générale de MITRA 15/30

■ Software :

- Assembleur MITRAS I; assembleur étendu MITRAS II; macro-générateur; LP 15; BASIC; FORTRAN IV; Bibliothécaire; Système de Gestion de Fichiers.

- 3 moniteurs : Moniteur de Base MOB; Moniteur Temps Réel MTR; Moniteur Temps Réel Disque MTRD.

■ Périphériques :

• Les périphériques Gamme I (du modèle 20) :

- Téléscriptrice de service (avec lecteur-perforateur de ruban)
- Lecteur de ruban rapide 300 car/s.
- Perforateur de ruban 60 car/s.
- Machine à écrire de logging 15 car/s.
- 2 à 128 lignes d'entrées numériques de 16 bits à niveau logique ou filtrées, ou à relais
- 2 à 64 lignes de sorties numériques de 16 bits à niveau logique, ou à relais
- Entrées de comptage; horloge temps réel
- Entrées analogiques.

• Les périphériques Gamme II :

- Disques à têtes fixes; temps d'accès moyen 10 ms, cadence de transfert 150 ko/s; capacité 100 ko à 1 600 ko.
- Disques amovibles à têtes mobiles, temps d'accès moyen 60 ou 90 ms, cadences de transfert 100 ou 150 ko/s.; capacités 2,5 à 5 Mo ou 6,2 à 24,8 Mo.
- Lecteurs de cartes 300 et 600 cpm
- Perforateur de cartes 20 ou 40 col/s.
- Imprimantes 132 col, 200, 400 et 600 lpm
- Dérouleurs de bandes magnétiques 800/1600 bpi, 20 ou 40 ko/s.
- Coupleur de transmission pour 1 ligne synchrone, Full duplex 1200/4800 bauds
- Coupleur de transmission pour 2 lignes asynchrones, Full duplex 50 à 1200 bauds, télégraphique ou téléphonique.

• Les périphériques Gamme III :

- Interface OCTET permettant la connexion des périphériques des séries CII 10 000 ou IRIS : Lecteurs de cartes, Imprimantes, Dérouleurs de bandes magnétiques, etc...

I-4. SYSTEME D'EXPLOITATION MITRA 15

Suivant qu'on dispose ou non d'un disque rapide, on distingue deux versions du software : système résident et système disque.

■ Le système résident permet de disposer :

- de deux moniteurs : le Moniteur de Base (MOB) et le Moniteur Temps Réel (MTR)
- des assembleurs MITRAS et LP 15, des compilateurs BASIC et FORTRAN IV et d'un macro-générateur.

■ Le système disque permet de disposer :

- du Moniteur Temps Réel Disque MTRD
- des processeurs du système résident, d'un bibliothécaire et d'un module d'enchaînement.

Le software comprend en outre :

- des commandes d'aide à la mise au point disponibles en extension de chaque moniteur.
- une bibliothèque complète de programmes mathématiques "temps réel", de transmission, et d'un système de gestion de fichiers.
- des simulateurs sur ordinateurs CII 10 070, IRIS 50, IRIS 80, IBM 360...

	4 K mots	8 K mots	12 K mots	16 K mots	simulation
exploitation système résident	moniteur de base MOB	moniteur temps réel MTR			analyseur de commandes interpréteur
système disque		moniteur temps réel disque MTRD	module d'enchaînement		
production de programmes système résident	MITRAS 1 éditeur de liens chargeur-éditeur	MITRAS 2 BASIC	LP 15	FORTRAN IV	assembleur éditeur de liens
système disque		MITRAS 2 éditeur de liens BASIC LP 15 bibliothécaire	FORTRAN IV	macro-générateur	LP 15
bibliothèque	bibliothèque mathématique bibliothèque temps réel bibliothèque de transmission système de gestion de fichiers packages				

Structure du software Standard MITRA 15

Assembleur MITRAS I

Programme de traduction du langage symbolique MITRAS produisant en une seule passe un programme objet en binaire translatable, une liste objet et une liste des erreurs détectées.

Le programme source et le binaire translatable sont normalement supportés par du ruban perforé; nécessite 4 K mots de mémoire.

Assembleur Etendu MITRAS II

Programme de traduction du langage symbolique MITRAS disposant d'un plus grand nombre de directives que l'Assembleur MITRAS I, nécessite 8 Kmots de mémoire.

EDITEUR DE LIENS

Programme fournissant en deux passes à partir de programmes en binaire translatable, issus de différents assemblages ou compilations, un module au format image mémoire translatable pouvant être chargé pour exécution par le moniteur de base.

L'éditeur de liens produit également une carte de l'implantation relative des différents modules et la liste des noms des sous-programmes communs appelés; nécessite 4 Kmots de mémoire.

MONITEUR DE BASE MOB

Assure le contrôle de l'ordinateur et permet à l'utilisateur de communiquer avec le système et les différents processeurs de base. Ses principales fonctions sont :

- traitement des déroutements
- contrôle des interruptions internes
- chargement des programmes
- contrôle des entrées/sorties
- contrôle de l'exécution des programmes.

Nécessite 4 Kmots de mémoire.

MONITEUR TEMPS REEL MTR

Assure la gestion simultanée en mémoire de travaux liés à des interruptions et de travaux du type Centre de Calcul.

Effectue et contrôle toutes les opérations privilégiées telles que la mise en oeuvre des entrées/sorties ou la protection mémoire, et assure le dialogue avec l'opérateur; nécessite 8 Kmots de mémoire.

MONITEUR TEMPS REEL DISQUE MTRD

Version disque du moniteur MTR, permet en outre la gestion des structures de recouvrement (overlays), gère les bibliothèques "système" ou "utilisateur" et assure l'enchaînement automatique des programme de Centre de Calcul (compile-link - load and Go); nécessite 8 Kmots de mémoire et un disque rapide.

CHARGEUR - EDITEUR DE LIENS

Programme permettant de charger en une passe des programmes au format binaire translatable pour exécution immédiate.

Ce processeur n'accepte que le binaire produit par l'assembleur MITRAS 1; nécessite 4 K mots de mémoire.

LP 15

Langage de type Assembleur possédant une syntaxe qui se rapproche de celle d'un langage évolué comme ALGOL, mais offrant la particularité de pouvoir accéder directement aux registres de MITRA 15.

Le binaire objet produit est pratiquement aussi performant que celui qui est obtenu avec un langage d'assemblage; nécessite 12 K mots de mémoire sans disque.

BASIC

Compilateur conversationnel permettant une exploitation en time sharing et autorisant le traitement des données alphanumériques; nécessite 8 K mots de mémoire.

FORTRAN IV

Compilateur une passe produisant un programme objet en binaire translatable au format admissible par l'éditeur de liens.

Permet de faire appel à des sous-programmes écrits dans un autre langage et mis en format binaire translatable; compatible 10 020, IRIS 45 et IRIS 50; nécessite 16 K mots de mémoire sans disque, ou 12 K mots avec disque.

EXTENSION AMAP

Chaque moniteur possède une extension AMAP d'aide à la mise au point permettant les traces d'exécution, les arrêts sur adresse, les dumps et modifications mémoire à l'aide de commandes moniteur spécialisées à cet effet.

MODULE D'ENCHAINEMENT

Permet d'enchainer automatiquement un train de travaux dans la zone de traitement différé en simultanéité avec des programmes temps réel.

Ce processeur fonctionne sous contrôle du moniteur temps réel disque MTRD; nécessite 12 K mots de mémoire et un disque rapide.

MACRO-GENERATEUR

Programme de traduction de procédures définies par l'utilisateur, produisant en une passe un programme en langage d'assemblage ou de compilation; nécessite 16 K mots de mémoire.

BIBLIOTHECAIRE

Permet de gérer le fichier constitutif de la bibliothèque du système à l'aide d'ordres d'insertion, d'effacement, de remplacement, de copie, de chargement, de vidage sur support externe...; nécessite 8 K mots de mémoire et un disque rapide.

PROGRAMMES DE SERVICE

Programmes permettant :

- la mise à jour et la correction de programmes source enregistrés sur support à accès séquentiel (ruban perforé, bande magnétique, etc...);

- la manipulation et la mise à jour des programmes d'une bibliothèque enregistrés sur support séquentiel.

Nécessite 4 K mots de mémoire.

SIMULATEURS MITRA 15

Programmes de simulation exploitables sur 10 070, IRIS 50, IRIS 80, IBM 360... comportant :

- un interpréteur de MITRA 15
- l'assembleur MITRAS et l'éditeur de liens
- un générateur de système
- le compilateur LP 15.

Ils permettent d'assembler, d'éditer et de mettre au point des programmes qui pourront être exploités sur toutes les configurations MITRA 15.

1-5. APPLICATIONS

LABORATOIRES	Spectrométrie, Chromatrographie gazeuse, Cristallographie...
MEDECINE	Analyses Chimiques, Electrocardiographie...
ENGINEERING	Test de composants, Séismographie, Télémétrie...
INDUSTRIE Surveillance, Automatisme, Contrôle de Processus	Chimie, Pétro-chimie, Sidérurgie, Mécanique, Aéronautique...
TELEINFORMATIQUE Ordinateur frontaux, Satellites, Concentrateurs-diffuseurs	Entreprises décentralisées, Administrations, Universités...
CALCUL SCIENTIFIQUE Time Sharing, Centre de traitement	Enseignement, Bureau d'Etudes, Entreprises...
GESTION DE TRANSACTIONS Saisie d'informations, Gestion de fichiers	Assurances, Banques, Administrations...

2. Structure générale de l'ordinateur

L'ordinateur MITRA 15 est bâti autour d'une mémoire principale unique à architecture plane, modulaire par bloc de 4 K mots de 16 bits, et disposant de quatre accès permettant la connexion de une à quatre unités de traitement ou coupleur d'accès direct à la mémoire.

Les unités de traitement comportent des mémoires microprogrammées ROM (Read Only Memory); selon la nature des microprogrammes implantés dans ces mémoires les unités de traitement peuvent remplir les fonctions suivantes :

- Unité Centrale
- Unité d'Echange
- Unité spéciale adaptée à un traitement particulier.

Chaque unité de traitement pilote un bus périphérique le "Minibus" sur lequel sont directement connectés les organes périphériques.

II-1. MEMOIRE PRINCIPALE A TORES DE FERRITE

L'unité élémentaire d'information de la mémoire à tores est le mot.

Chaque mot comprend 18 chiffres binaires dont 16 d'information, 1 pour le contrôle de parité et 1 pour la protection mémoire.

La mémoire fonctionne par demi-cycles séparés. Pour lire un mot il faut ainsi effectuer un demi-cycle de lecture avec effacement suivi d'un demi-cycle de réécriture. Pour écrire un mot, il faut, après un demi-cycle d'effacement, effectuer un demi-cycle d'écriture.

Le temps d'accès de la mémoire est 400 ns (1/2 cycle), et un cycle de lecture/écriture est 800 ns.

Bien que les échanges avec la mémoire s'effectuent en fait par mots, les micro-commandes permettent au programmeur d'opérer par octets, c'est-à-dire par demi-mots. Toutes les adresses de MITRA 15 sont des adresses d'octets et les mots sont à des adresses paires.

La mémoire est modulaire par blocs de 4096 mots, c'est-à-dire 8 192 octets. L'ordinateur MITRA 15 peut avoir huit blocs au maximum, soit 32 768 mots (ou 65 536 octets).

La logique de commande fournit d'une part les signaux nécessaires au fonctionnement technologique de la mémoire (commande des demi-cycles proprement dits), d'autre part, les signaux de transfert pour les échanges avec les unités de traitement; enfin elle gère les priorités respectives des quatre accès.

II-2. UNITE DE TRAITEMENT

Le fonctionnement d'une Unité de Traitement de MITRA 15, et plus particulièrement celui de l'Unité Centrale, peut être décrit suivant deux niveaux totalement distincts :

■ Le premier niveau peut être qualifié de niveau utilisateur, c'est le seul qu'il soit nécessaire de connaître pour programmer une application sur l'ordinateur MITRA 15.

Il comprend :

- le jeu d'instructions standard qui est détaillé dans le chapitre "instructions"
- les six registres généraux du bloc zéro
- les cinq indicateurs programme
- le système d'interruption.

■ Le deuxième niveau correspond à ce que l'on peut qualifier de niveau de la micro-machine.

Cette micro-machine comprend les éléments suivants :

- un jeu d'une quarantaine de micro-instructions élémentaires réalisées par hardware.
- une mémoire constituée par les cartes de mémoire morte à lecture seulement et qui contient l'ensemble des sous-programmes, que nous appelerons plutôt microprogrammes, définissant le jeu d'instructions standard de MITRA 15 et les fonctions de couplage des périphériques.
- des registres de travail
- des indicateurs d'état de la micro-machine
- un système dit de suspension qui joue à ce niveau le même rôle que joue le système d'interruption au niveau 1.

Nous décrivons dans ce qui suit les différents éléments constitutifs d'une unité de traitement :

- la mémoire morte de microprogrammation
- les registres S et M de liaison avec la mémoire centrale
- les blocs registres rapides
- les indicateurs
- les systèmes d'interruption et de suspension.

II-3. MEMOIRE MICROPROGRAMMEE ROM

La mémoire ROM est une mémoire à lecture non destructive qui est préenregistrée à la construction, et réalisée en circuits intégrés (temps d'accès de 60 nanosecondes pour un mot).

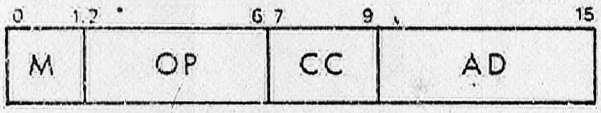
Chaque mot de cette mémoire comprend 16 chiffres binaires et représente une micro-instruction.

La mémoire de commande peut comporter 512 ou 1 024 mots par unité de traitement.

Toutes les micro-instructions s'exécutent en 300 nanosecondes.

L'adresse de la micro-instruction en cours d'exécution est fournie par le registre T de dix positions binaires.

Le format d'une micro-instruction est le suivant :



Chaque micro-instruction a un double rôle :

1) Commander certaines fonctions :

- Commande de la mémoire (deux chiffres binaires: zone M)

- Code opératoire principal (cinq chiffres binaires dans le cas général : zone OP)

- Code complémentaire (trois chiffres binaires : zone CC) qui donne par exemple l'adresse d'un registre général.

2) Définir l'adresse de la micro-instruction suivante (par un déplacement de six positions binaires : zone AD) c'est-à-dire définir la nouvelle valeur de T.

En effet, les micro-instructions ne sont pas rangées en séquence.

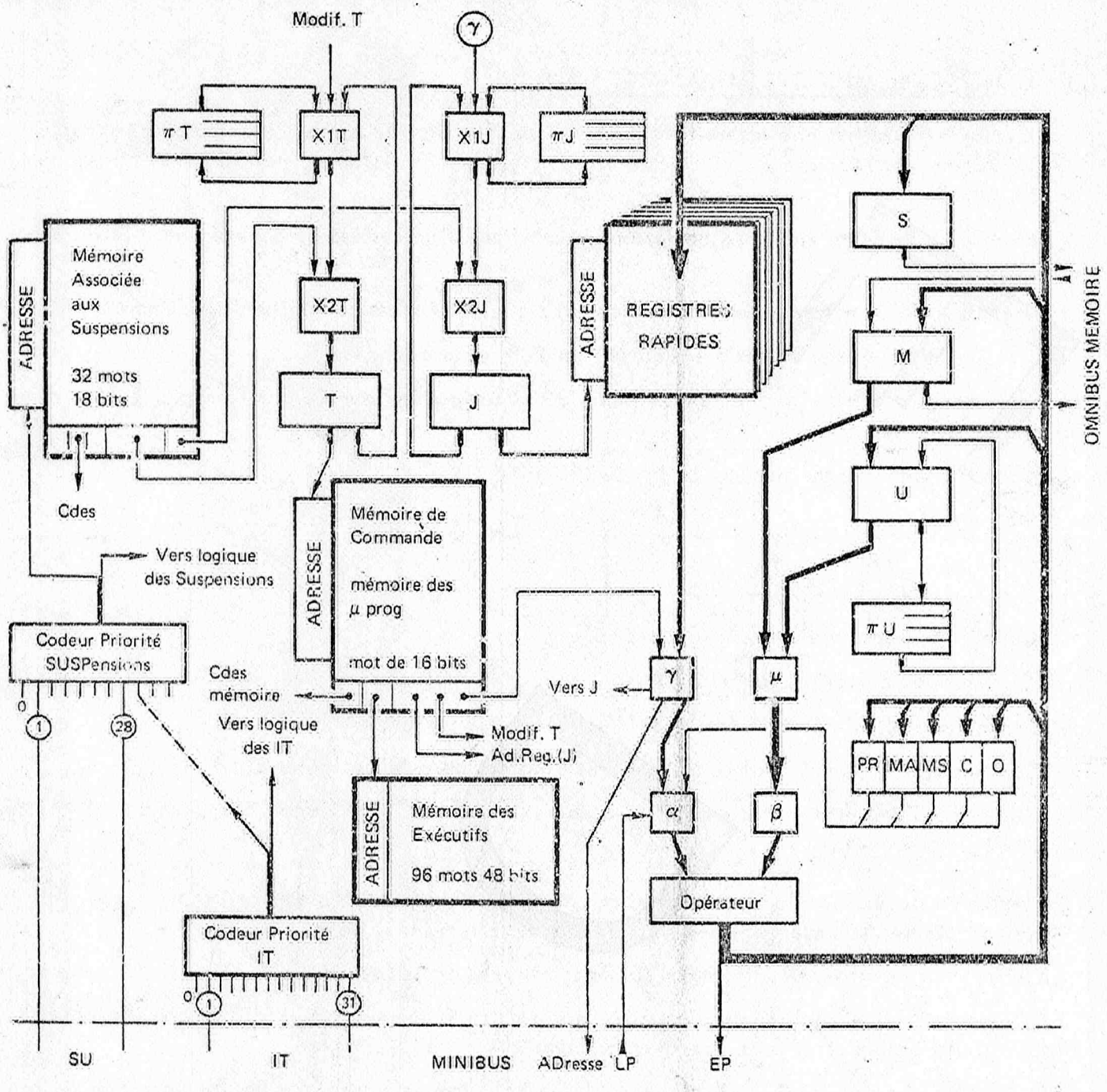
Il n'y a donc pas d'additionneur associé au registre T puisque celui-ci n'a pas à progresser d'une unité pour passer à la micro-instruction suivante.

Ce qui différencie Unité Centrale et Unité d'Echange est la nature des mémoires ROM qui y sont implantées.

Dans l'Unité Centrale : la mémoire MC1 exécute le code d'instruction de base et les fonctions de couplage des périphériques uniquement connectables sur le Minibus de l'Unité Centrale (Gamme I).

La mémoire MC2 exécute le code complémentaire (instructions optionnelles) et les fonctions de couplage des périphériques connectables sur le Minibus de l'Unité Centrale, ou le Minibus de l'Unité d'Echange (Gamme II).

La mémoire MC3 exécute les fonctions de comptage des périphériques uniquement connectables sur le Minibus d'une Unité d'Echange (Gamme III).



II-4. REGISTRES

II-4.1. Registres de liaison avec la mémoire

- le registre d'adresse S est de 15 positions binaires bien que les adresses en aient 16. Le plus faible poids d'une adresse, qui fixe l'octet à prendre en compte à l'intérieur du mot adressé, est en effet ignoré par la logique de la mémoire.

- le registre des données M possède 18 positions binaires comme les mots de la mémoire. Deux de ces positions servent aux tests de parité et de protection. Les 16 autres permettent d'effectuer des échanges de données avec le registre U.

II-4.2. Registres rapides

Une Unité de Traitement MITRA 15 comprend en standard huit blocs de huit registres de 16 positions binaires réalisés en circuits intégrés, numérotés de 0 à 7; huit blocs complémentaires sont disponibles en option.

Ce qui différencie les registres de l'Unité Centrale et ceux de l'Unité d'Echange est leur affectation.

Dans l'Unité Centrale, le premier bloc, ou bloc 0, est affecté à l'exécution des programmes. Il comprend les six registres :

P - compteur ordinal

L - registre de la base locale

G - registre de la base générale

A - accumulateur

E - extension de l'accumulateur

X - registre d'index

ainsi que les registres V et W utilisés par les microprogrammes.

Les autres blocs sont normalement affectés au traitement des échanges par suspension avec les périphériques (mémoires de voies).

Dans l'Unité d'Echange tous les blocs sont affectés au traitement des échanges avec les périphériques.

Chaque registre possède une adresse comprise entre 0 et 63 (ou 127).

Dans les microprogrammes l'adresse d'un registre général est élaborée à partir :

- de la zone de la macro-instruction (trois positions binaires) prévue à cet effet.
- et du contenu du registre J.

Nous verrons au paragraphe II-8., que dans le cas d'une interruption rapide, celle-ci provoque une commutation automatique de bloc registre.

Le nouveau bloc joue alors le même rôle que le bloc zéro pour les autres programmes.

II-5. UNITE ARITHMETIQUE ET LOGIQUE

L'unité arithmétique et logique est composée du registre universel U et d'un opérateur à deux entrées. Le registre U possède 16 positions binaires, il n'est pas accessible directement aux instructions mais sert de registre accumulateur à la "micro-machine".

A ce titre, il peut contenir un des opérandes de la micro-instruction et/ou en recueillir le résultat.

Les deux opérandes de la micro-instruction peuvent également être issus :

- d'un registre général (opérande 2)
- du registre M de la liaison avec la mémoire (opérande 1)

- de l'interface d'Entrée/Sortie (opérande 2)
- de la mémoire de commande (opérande 2)
- de la pile (opérande 2)
- des indicateurs (opérande 2).

Les résultats de l'opération sont rangés dans les éléments suivants :

U registre universel

M, S } registres de liaison avec la mémoire

Registre général

Indicateurs.

II-6. INDICATEURS

L'Unité Centrale de MITRA 15 possède neuf indicateurs :

■ Quatre indicateurs sont utilisés seulement par la micro-machine.

B : associée aux débordements de U

TN : nullité du résultat de la micro-instruction

TO : signe du résultat de la micro-instruction

AO : adresse de l'octet manipulé

■ Cinq indicateurs sont accessibles au programme :

C = Carry

Cet indicateur a deux sens différents suivant l'instruction qui le positionne.

• Report (Instruction de type arithmétique)

- Si on additionne un nombre positif (soustrait un nombre négatif), et que le résultat est atteint sans passer par zéro, Carry est positionné à zéro.

- Si on additionne un nombre positif (soustrait un nombre négatif), et que le résultat est atteint en passant par zéro, Carry est positionné à un.

- Si on additionne un nombre négatif (soustrait un nombre positif) et que le résultat est atteint sans passer par zéro, Carry est positionné à un.

- Si on additionne un nombre négatif (soustrait un nombre positif) et que le résultat est atteint en passant par zéro, Carry est positionné à zéro.

• Pour les autres instructions positionnant Carry, celui-ci indique après exécution si la valeur d'un registre est nulle ou, dans le cas d'une comparaison, si les deux valeurs sont égales.

O = Overflow

De même que Carry, cet indicateur a deux sens différents suivant la dernière instruction exécutée.

• Instruction de type arithmétique, Overflow a le sens d'un débordement. Plus précisément quand les deux opérandes sont de même signe et que le résultat est de signe contraire, Overflow est positionné à 1. Dans tous les autres cas il est positionné à zéro.

• Pour les autres instructions positionnant Overflow, celui-ci indique après exécution, si la valeur dans un registre est négatif (bit zéro à 1) ou, dans le cas d'une comparaison, si le contenu du registre A est inférieur au contenu du mot adressé.

MS = Indicateur de Mode

Cet indicateur est égal à 1, lorsque le programme doit s'exécuter en mode Maitre.

Il est à zéro, dans le cas du mode normal ou esclave.

Dans le chapitre "Instructions", le positionnement de ces trois indicateurs, est précisé pour chaque instruction.

MA = Indicateur masque des interruptions

Cet indicateur est à 1 pour des interruptions masquées, sinon il est à zéro.

PR : Clés de Protection

Suivant la valeur de PR (1 ou 0) le programme peut accéder à toute la mémoire, ou seulement aux mémoires dont le verrou de protection a la valeur zéro.

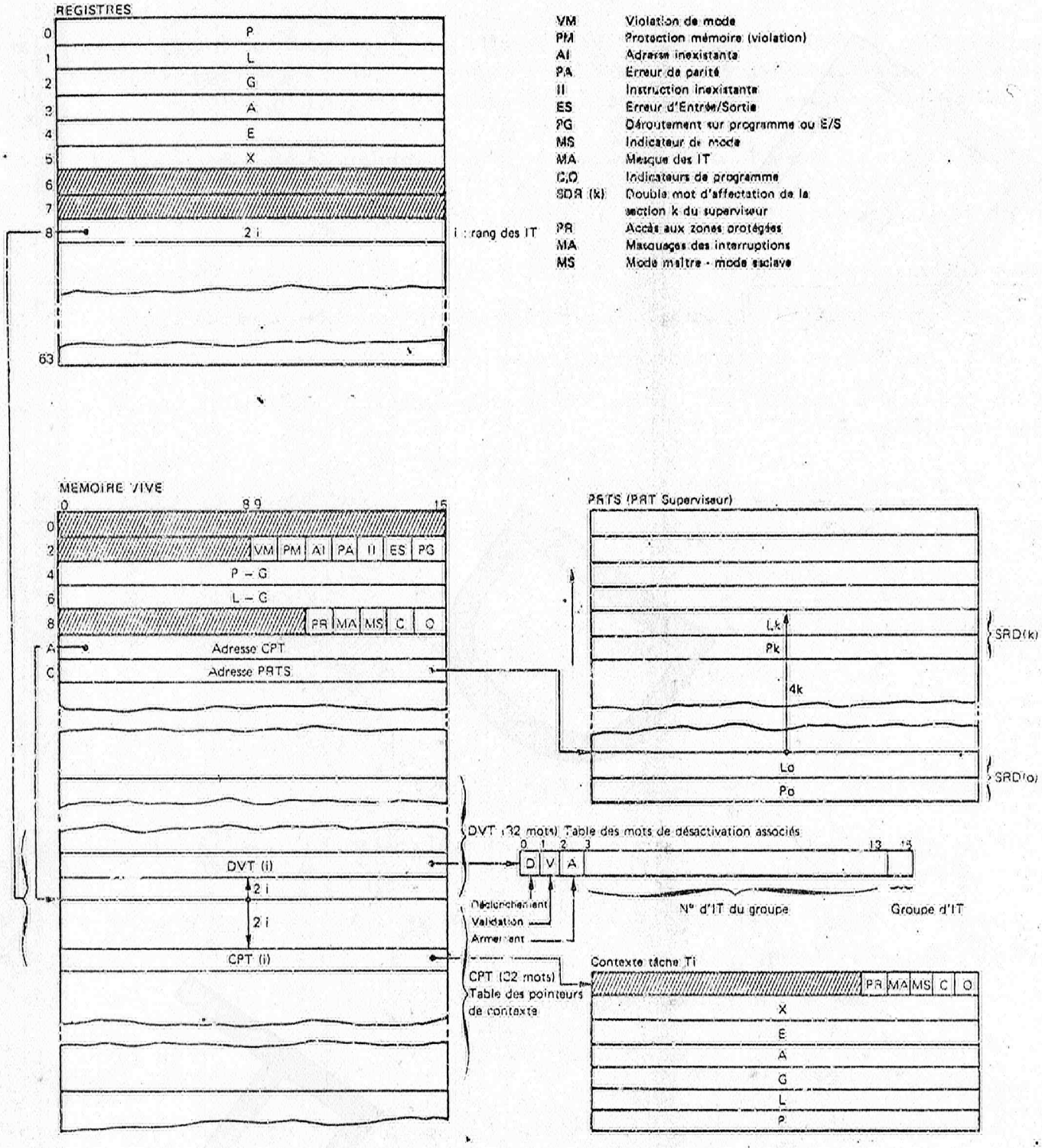
Ces cinq indicateurs font partie du contexte d'un programme.

II-7. COMMUNICATION AVEC L'ENVIRONNEMENT

L'Unité de Traitement est reliée aux coupleurs de périphériques par un bus accessible par micro-instructions appelé Minibus. L'interface comporte :

- pour les données : 16 positions binaires à l'émission et 16 à la réception.
- trois positions de fonction
- pour les adresses : 6 positions binaires ou 10 dans certains cas.
- une synchronisation
- une remise à zéro.

Le Minibus périphérique sur lequel se connectent les coupleurs des périphériques, comporte 16 bits uni-directionnels de lecture et d'écriture, un bus d'adresses et de commandes de périphériques et les fils d'appels d'interruptions et de suspensions.



Liaisons avec la micromachine

II-8. INTERRUPTIONS - SUSPENSIONS - DEROUTEMENTS

II-8.1. Interruptions

Le système d'interruption est mis en oeuvre quand :

- un signal d'interruption apparaît
- une micro-instruction spéciale, située par définition aux "points interruptibles " apparaît.
- les interruptions ne sont pas masquées
- le niveau de programme en cours est inférieur à celui de l'interruption incidente.

Le nombre de niveaux d'interruptions est égal à 32.

A chacun des 32 niveaux, est attachée une adresse mémoire qui contient un pointeur de contexte, désignant un programme spécifique de ce niveau.

Les 32 pointeurs de contexte sont situés dans une table pointée par le contenu du mot d'adresse 10.

Lorsqu'une interruption apparaît :

- elle est mémorisée dans une bascule (une bascule par signal)
- son niveau est codé par Hardware et comparé à celui de la tâche en cours, qui est présent dans la machine (registre 8).
- si l'interruption est prise en compte, son niveau (de 0 à 31) est rangé par Hardware dans la micro-machine au moment de l'exécution de la micro-instruction test interruption.
- on effectue alors par microprogramme les opérations suivantes :

. rangement du contexte de la tâche interrompue à une adresse fonction de son rang (ce dernier est rangé dans le registre rapide d'adresse 8).

. appel du contexte de la tâche interrompante, en attente à une adresse fonction de son rang.

. appel de la première instruction de la tâche appelante.

(Voir figure "Liaisons avec la micro-machine" page II-8).

Lorsque la tâche correspondante est terminée ou doit se mettre en attente d'un évènement, elle libère l'unité de traitement par une instruction de désactivation d'interruption et changement de contexte DIT dont le fonctionnement est :

. acquittement de l'interruption qui a provoqué l'appel de la tâche

. rangement du contexte de la tâche

. appel d'une tâche de même niveau, en attente, ou à défaut d'une tâche de niveau immédiatement inférieur. S'il n'y en a pas l'ordinateur tourne sur une boucle d'attente d'un évènement extérieur au niveau le plus bas.

Le nombre total de niveaux d'interruptions est 32, 4 internes et 28 externes. Les interruptions peuvent d'autre part être regroupées par 4 sur un même niveau, portant le nombre total d'interruptions externes possibles à 28 x 4 = 112.

Les coupleurs des périphériques standard utilisent généralement un niveau d'interruption par coupleur.

Les niveaux d'interruptions internes sont affectés aux tâches suivantes :

- Contrôle pupitre
- Mise sous tension
- Mise hors tension
- Programme (niveau 0)

Interruption rapide

L'un des niveaux d'IT externes peut être, en option, de catégorie "rapide", c'est-à-dire qu'elle appelle une tâche dont le contexte est présent dans un bloc registre différent du bloc 0 qui contient le contexte de la tâche interrompue. Dès lors, le changement de tâche ne nécessite que le rangement des indicateurs dans le bloc 0. Il est effectué en 2 µs environ.

En fin d'exécution de cette tâche "rapide", le retour à la tâche interrompue (dont le contexte est toujours présent dans le bloc 0) se fait par une instruction spéciale DITR qui n'effectue pas le changement de contexte habituel dans le bloc 0.

II-8.2. Suspensions

Une suspension est une demande d'arrêt du déroulement du microprogramme en cours, et qui provoque la mise en jeu d'un microprogramme spécial. La demande de suspension peut venir d'un périphérique ou peut être interne à la micro-machine (unité de traitement).

Lorsqu'il y a suspension, l'état de le micro-machine, c'est-à-dire le contenu des registres U, J, T, des indicateurs B, Tz, To, Ao est rangé dans une pile. Le microprogramme de suspension se déroule alors.

En fin de suspension, il y a restitution dans les registres U, J, T .... des valeurs qui avaient été précédemment rangées dans la pile.

La hauteur de la pile est égale à 4. Ce qui veut dire que le nombre de niveaux de suspension est égal à 4. Le nombre total de signaux de suspension est égal à 32, soit 8 par niveau :

- cinq suspensions internes :

. déroutements (1)
. interruptions (2)
. panneau commande (1)
. protection secteur (1)

- 27 suspensions externes, associées aux périphériques.

II-8.3. Déroutements

Un déroutement a pour origine un incident détecté à la fin d'une micro-instruction.

Le micro-programme de déroutement :

- Protège dans les octets 4 à 9 de la mémoire, le contenu des registres L, P et des indica-

teurs au moment de l'exécution de l'instruction ayant provoqué le déroutement.

- Indique par un bit du mot 2 de la mémoire qu'elle est la cause de déroutement.

- Réalise l'appel de la section zéro du Superviseur.

Les incidents pouvant provoquer un déroutement standard sont les suivants :

- Adresse mémoire inexistante : l'utilisateur a indiqué une adresse qui déborde la mémoire réelle dont il dispose.

- Défaut de protection mémoire : l'utilisateur tente d'écrire dans une zone de mémoire protégée alors que la clé PR est à zéro.

- Défaut de parité sur mémoire ferrite.

Les autres déroutements possibles peuvent être provoqués par :

- Violation du mode de fonctionnement : l'utilisateur utilise des instructions privilégiées, alors que son programme est écrit en mode esclave.

- Instruction inexistante : erreur dans le code de l'instruction.

- Watch Dog.

Dans tous les cas :

- l'instruction en cours n'est pas achevée

- la pile de rangement de la micro-machine n'est pas mise en oeuvre

- un microprogramme spécial crée un appel au superviseur.

Le traitement réalisé par les moniteurs standard est décrit dans les notices d'utilisation correspondantes.

Le mot d'état du déroutement est décrit dans la figure "Liaisons avec la micro-machine" page II-8.

II-9. MODE ET PROTECTION.

II-9.1. Modes de fonctionnement

- Mode normal ou esclave.

Dans ce mode les instructions privilégiées ne peuvent être exécutées. Tout essai pour en exécuter une, provoque un déroutement de "violation de mode", l'indicateur MS est à zéro.

- Mode privilégié ou maître.

Ce mode permet l'éxécution de toutes les instructions, qu'elles soient privilégiées ou non.

L'indicateur MS est à 1.

Les différents modules du moniteur seront par exemple des programmes qui s'exécuteront obligatoirement en mode Maître (cf. instructions CSV et RSV).

Signalons que le traitement des modes d'adressage est différent en mode maître et en mode esclave (cf. chapitre "modes d'adressage") pour permettre en mode maître d'accéder à des adresses absolues.

II-8.5. Système de protection mémoire

Le système de protection est mis en service par la clé PM du panneau de commande.

Ce système fonctionne de la façon suivante :

- A chaque mot mémoire est associé un "verrou" de protection de un bit qui peut être positionné par l'instruction LDP (Load Protection).

- L'état programme comporte un indicateur PR dont la fonction est celle d'une "clé".

Si la clé a la valeur 1 (passe-partout) le programme peut accéder à toute la mémoire.

Si la clé a la valeur 0 le programme ne peut accéder qu'aux mémoires dont le verrou a également la valeur 0.

Chargement de la clé PR

L'indicateur PR est chargé avec le contexte du programme.

Il est préservé avant d'être forcé à 1 lors de tout appel au superviseur et restitué lorsque le superviseur rend le contrôle au programme appelant.

Violation de protection

Si un programme dont la clé est "nulle" tente d'accéder à une mémoire dont le verrou est égal à 1, le système de protection se déclenche et provoque un déroutement "violation de protection".

Protection mémoire et mode de fonctionnement sont indépendants.

3. Structure d'un programme

III-1. QU'EST-CE QUE LA MODULARITE ?

En programmation, comme dans toute autre technique, la modularité consiste à décomposer les systèmes en petits éléments à interfaces normalisés.

Depuis l'utilisation de la notion de "sous-programme" la modularité est un fait acquis en programmation. Un programme principal pouvant également être un module nous emploierons de préférence le terme "section".

Les avantages de la modularité sont notamment de :

- faciliter la spécification d'un système.
- faciliter sa réalisation en parallèle par plusieurs programmeurs.
- permettre la réutilisation de sections identiques d'un système à l'autre.
- faciliter la mise au point et le suivi d'un produit.

III-2. QU'EST-CE QU'UNE SECTION ?

Une section est principalement composée d'une "suite d'instructions" que l'on appelle segment de programme. Ces instructions ont pour objet le traitement de "données" qui peuvent être propres à une section ou partagées entre plusieurs sections.

Les données propres à une section forment un segment de données locales.

Les données communes forment une section de données communes.

Une section est donc soit la section de données communes (CDS : Common Data Section) soit l'ensemble d'un segment de données locales (LDS : Local Data Segment) et d'un segment de programme exécutable (LPS : Local Program Segment).

Remarque :

Dans la suite du texte, l'expression sous-section pourra être substituée au terme segement (sous-section de données : LDS, sous-section de programme exécutable : LPS).

- On accède à la CDS à partir de tout le programme.

En particulier, on accédera à la CDS à partir d'un LDS en mode général (direct, indirect ou indirect indexé).

Les symboles et étiquettes définis dans la CDS sont communs à tout le programme.

- On accède à un LDS à partir du LPS associé en mode local (direct, indirect, ou indirect indexé).

Les symboles et étiquettes définis dans un LDS sont locaux à la section. Ils peuvent néanmoins être référencés en CDS.

- Un segment de programme ne comporte que des éléments non modifiables (instructions) ce qui facilite la translatabilité et l'écriture de sous-programmes réentrants.

III-3. BASES DES SECTIONS ET SEGMENTS

• Base Générale G

La base générale G est associée de façon bi-univoque au programme ; c'est la base implicite à laquelle toutes les adresses référencées par le programme sont relatives. Elle est donc ajoutée automatiquement par la micro-machine aux adresses définies dans les instructions.

• Base L

La base L est la base locale implicite ou base des données locales, associée à un segment de données locales.

• Base P

La base P est la base programme associé à un segment de programme. La base P contient initialement l'adresse de début d'exécution de la section, puis constitue le compteur ordinal en cours d'exécution de la section.

La valeur effective des bases L et P d'une section peuvent être ignorées du programmeur au moment de l'écriture du programme. Elles sont définies automatiquement par l'éditeur de liens en valeur relative à la base générale du programme et stockées dans la PRT du programme (cf. paragraphe III-2)

III-4. CONSEQUENCES DE LA MODULARITE DES PROGRAMMES SUR MITRA 15

Du point de vue Hardware, la modularité implique l'existence d'instructions spéciales d'appel ou de retour de section.

En ce qui concerne la programmation, la modularité des programmes est une notion fondamentale du langage d'assemblage qui comporte des directives dites de sectionnement :

CDS : Common Data Section

LDS : Local Data Segment

LPS : Local Program Segment

IDS : Indirectement Data Segment.

FIN : Fin de segment ou de section (CDS, CDS, LPS, IDS)

On désigne par "module de programme" le résultat d'un assemblage ou d'une compilation. Dans le cas d'un module écrit en langage d'assemblage, on parlera de "module d'assemblage".

Chaque module d'assemblage doit se terminer obligatoirement par un directive END.

Un programme peut être constitué à partir de modules d'origines diverses tant par le langage source que par l'auteur ou la date de création.

Les différents modules sont reliés entre eux par l'éditeur de liens pour former le programme complet exécutable.

Remarque :

Pour faciliter l'écriture des programmes et notamment des sous-programmes réentrants, l'assembleur reconnait des "segments fictifs" images de données ultérieures ou de données appartenant à un autre LDS (ou CDS) que celui où la zone fictive est déclarée.

Ces segments fictifs jouent le rôle de paramètres formels, en particulier pour définir des déplacements relatifs au début de ces segments (description de blocs de données dynamiques, définition de valeurs d'index, etc...) mais n'engendrent aucun texte objet.

Exemple 1 : Organisation usuelle d'un programme

```
COMMON      CDS
TWB         RES     16
C1          DATA    1
            FIN

LOCAL       LDS
            RES     2
C2          DATA    &F0
            FIN

SPROG       LPS     LOCAL
DEB         LDA     = 3
            AND     C2
            RTS
            FIN     DEB

LOCP        LDS
            RES     2
U           DATA,1  28
V           DATA,1  31
TAB         DATA    ATAB
ATAB        RES     1024
            FIN

PRINC       LPS     LOCP
INIT        LDA     U
            ADD     = C1
            STA     TAB
            CLS     SPROG
            CSV     M:EXIT
            FIN     INIT
            END     PRINC
```

Section de données communes

Segment de données locales

Segment de programme

Section 1

Segment de données locales

Segment de programme

Section 2

Marque fin de fichier (%EOD sur carte et ruban perforé).

Exemple 2 : Diverses organisations spéciales possibles

```
PROG    CDS
        .
        .
        FIN

LPS1    LPS     PROG
        .
        .
        FIN
        END     LPS1
```

Ce LPS n'ayant aucun LDS associé ne pourra utiliser le mode d'adressage local, il ne pourra utiliser que le mode général.

```
        CDS
        .
        .
        FIN

LDS1    LDS
        .
        .
        FIN

LPS1    LPS     LDS1
        .
        .
        FIN

LPS2    LPS     LDS1
        .
        .
        FIN
        END     LPS1
```

Chacun de ces deux LPS sera associé au même LDS. Les symboles locaux ne seront effacés qu'à l'apparition du prochain LDS. Il n'en reste pas moins qu'il s'agit bien de deux sections (deux éléments du PRT). La valeur de la base L étant à l'origine la même pour les deux sections, ces sections ne devront pas s'appeler entre elles (CLS).

```
        CDS
        .
        .
        FIN

LDS1    LDS
        .
        .
        FIN

LDS2    LDS
        .
        .
        FIN

LPS2    LPS
        .
        .
        FIN
        END     LPS2
```

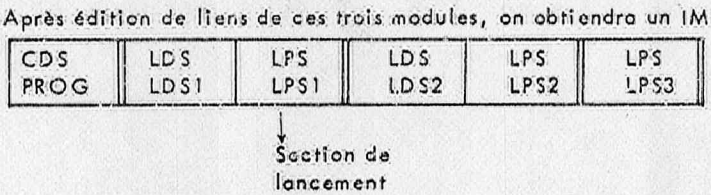
Ce LDS n'étant associé à aucun LPS, on ne peut y accéder que par indirection à travers un élément de CDS.
Cette utilisation est possible mais d'un faible intérêt.

Ce LPS ne peut faire référence au LDS de nom LDS2 puisque les symboles locaux sont perdus à chaque apparition d'une directive LDS.

Exemple 3 : Utilisation de segments ou sections fictives.

Premier module	Deuxième module	Troisième module	Remarques
PROG CDS RES 16 C1 DATA 2 C2 RES 4 C3 DATA D1 C4 RES 2 C5 DATA C2 FIN	PROG CDS DUM RES 16 C1 DATA 2 C2 RES 4 C3 DATA D1 C4 RES 2 C5 DATA C2 FIN	PROG CDS DUM RES 16 C1 DATA 2 C2 RES 4 C3 DATA D1 C4 RES 2 C5 DATA C2 FIN	Ces CDS sont images les unes des autres. Les CDS DUM fictives n'entraînent pas de génération de binaire. Elles serviront à satisfaire les modes d'adressage général et les références. Elles permettront également à chaque programme de disposer en clair des éléments qu'il manipule.
LDS1 LDS RES 2 D1 DATA C1 FIN LPS1 LDS LDS1 DEB1 LDA D1 CLS LPS2 CLS LPS3 CSV M:EXIT FIN DEB1 END LPS1			
	LDS2 LDS RES 2 D2 DATA 4 RES 5 D3 DATA C2 D4 DATA C4 FIN LPS2 LPS LDS2 DEB2 LDA D2 LDX =2 STA Ch,x RTS FIN DEB2 END	LDS2 LDS DUM RES 2 D2 DATA 4 RES 5 D3 DATA C2 D4 DATA C4 FIN LPS3 LPS LDS2 DEB3 LDA D4 LDX =0 STA C4 RTS FIN DEB3 END	Les deux segments de programme LPS2 et LPS3 sont connectés au même segment de données LDS2. Ils sont assemblés séparément, mais l'un des deux fait référence à un segment DUM (fictif) qui sert également à satisfaire les modes d'adressage local et permet au programmeur de disposer en clair des éléments qu'il manipule. Le segment DUM n'entraînera pas de génération de binaire.

Après édition de liens de ces trois modules, on obtiendra un IMT exécutable de la forme suivante :



III-5. ELEMENTS DE COMMUNICATION D'UN PROGRAMME

■ Bloc de travail (TWB : Task Working Block)

Les 16 premiers mots de la CDS forment le bloc de travail (TWB).

Cette zone de 16 mots est réservée au moniteur qui pourra y ranger l'adresse de retour dans la tâche appelante, ainsi que la base de données locales (L) de l'appelant et les indicateurs.

Le moniteur pourra y entretenir un pointeur vers la zone de données communes du système (ZC).

Il pourra utiliser la zone TWB comme mémoires de travail.

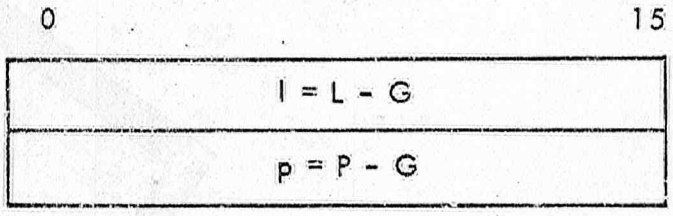
Les programmes qui font appel au moniteur doivent donc commencer leur CDS par une réservation de 16 mots.

C'est ce dispositif qui permet la réentrance des sections moniteur, celles-ci travaillant dans le programme appelant.

■ Table d'affectation des sections (PRT : Program Relocation Table)

- Une affectation de section est constituée par un double-mot d'affectation qui contient les valeurs initiales de L et de P relatives à G :

Double-mot d'affectation :
(SRD : Section Relocation Double-word).



- La PRT est formée de l'ensemble des SRD des sections constitutives du programme. Elle précède immédiatement l'adresse G, de sorte que l'adresse du SRD de la section n est égale à :

G - 4n

Cette table est constituée à l'édition de liens du programme.

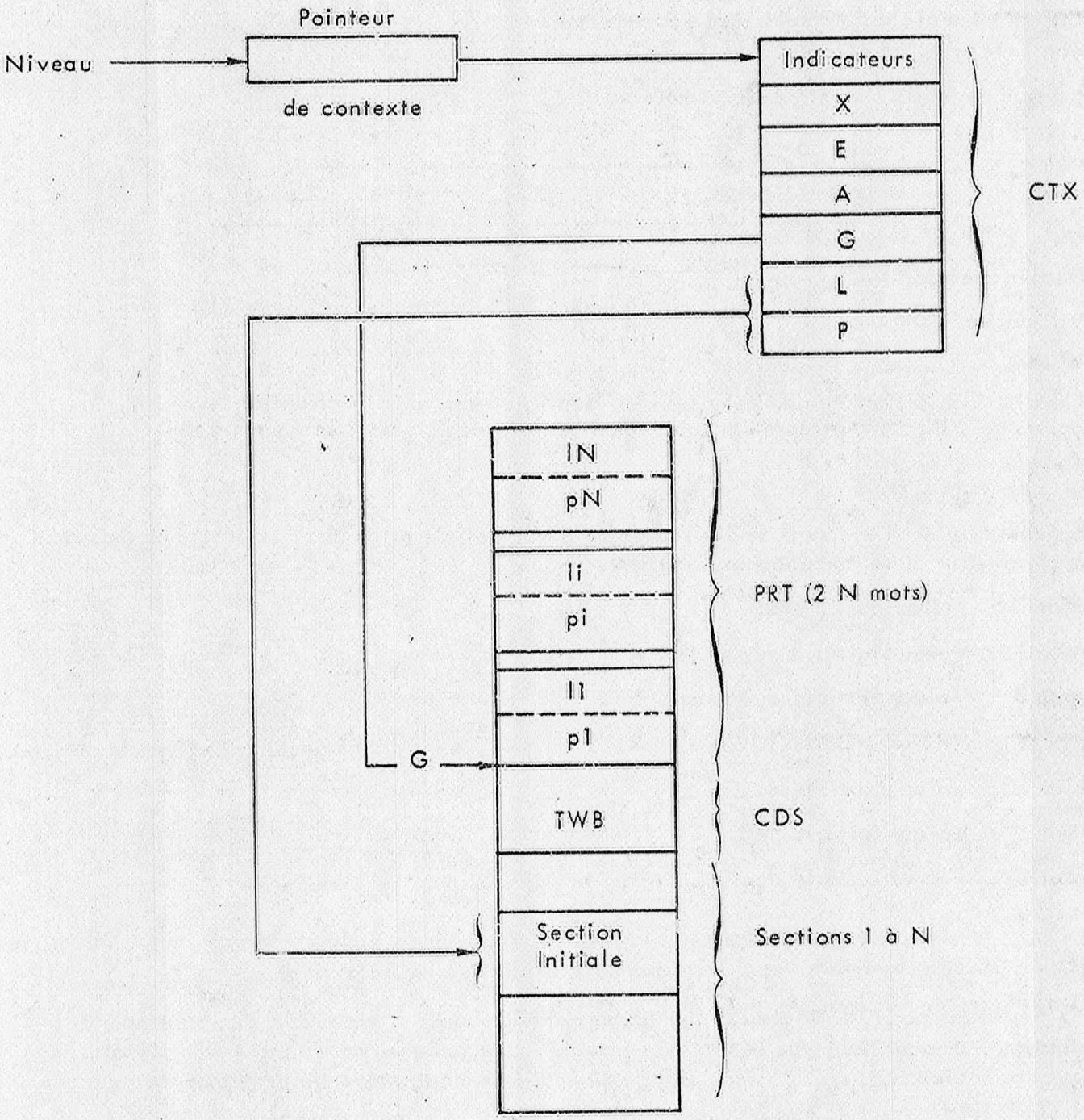
Nota :

L'adresse de la PRT du moniteur est définie par le contenu d'une adresse fixe connue, celle de la micro-machine (adresse 12).

La PRT constitue l'élément de liaison entre sections d'un même programme ou entre un programme et le moniteur (en ce qui concerne la PRT du moniteur).

Remarque :

La CDS étant accessible de n'importe quelle section constitue une liaison implicite entre les sections.



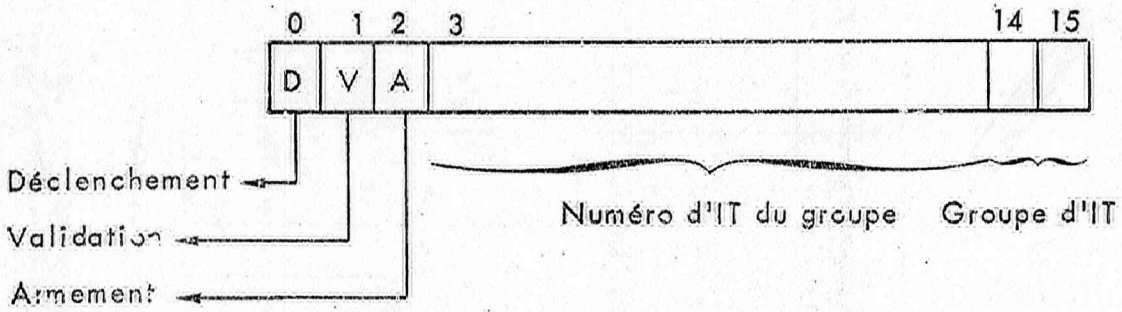
où li = Li - G et pi = Pi - G

Structure d'un programme

Table des mots de désactivation associés : DVT (DeVice Table)

C'est une table de 32 mots, qui précède dans la mémoire vive la table des pointeurs de contexte, également de 32 mots, CPT (Context Pointer Table).

Le mot de la DVT se présente comme suit :



Les bits 3 à 15 sont également appelés "configuration de l'interruption".
Le système d'interruption et la table DVT ont été décrits dans le chapitre II.

Contexte CTX

Le contexte est l'élément de liaison entre un niveau de priorité et le programme associé à ce niveau. Il se compose de 7 mots :

Mot 1 : Indicateurs d'état

Mot 2 : Valeur initiale du registre X

Mot 3 : Valeur initiale du registre E

Mot 4 : Valeur initiale du registre A

Mot 5 : Valeur initiale de G

Mot 6 : Valeur initiale de L

Mot 7 : Valeur initiale de P

Le contexte permet l'initialisation, la reprise d'une tâche et la protection de l'état d'une tâche, suivant que le niveau correspondant est activé ou désactivé.

En devenant actif, le niveau d'une tâche définit, dans la table CTX des pointeurs de contexe, un pointeur vers le contexe associé. Les registres P, L, G, A, E, X, ainsi que les indicateurs d'état, sont chargés à partir du contexte et le programme s'exécute à l'adresse P.

Inversement, lorsqu'un niveau est interrompu par un niveau plus prioritaire, ou lorsqu'il est acquitté, le contenu actuel des registres P, L, A, E, X et des indicateurs d'état sont rangés dans le contexte.

On se rapportera pour plus de détails à l'instruction DIT (chapitre Instructions).

III-6. APPELS DE SECTIONS

La section apparaît sous deux formes :

- section propre à un programme, qu'on atteint par le CALL SECTION (CLS)

- section disponible pour tous les programmes : section superviseur ou section de bibliothèque commune, qu'on atteint par le CALL SUPERVISEUR (CSV).

Appel d'une section

CALL SECTION (CLS)

A l'exécution d'une instruction CLS "appel section", la machine :

- Range les contenus de P (adresse programme) et L (base de zone de données locales) dans les deux premiers mots de la zone locale de la section appelée (après y avoir soustrait (G)). Ces éléments sont nécessaires au "retour" et doivent en effet être préservés.

- Charge P et L avec les adresses respectives du point d'entrée et du segment de données locales de la section appelée qui peut alors s'exécuter.

A l'exécution d'une instruction RTS "retour section" la machine :

- Restaure les registres P et L avec les valeurs qui avaient été préservées en début du segment local de la section appelée.

Nota :

Lorsque plusieurs sections d'un même programme, sont assemblées séparément et que l'on trouve dans l'une un appel à l'autre il n'est pas nécessaire de déclarer que la section appelée est externe au module. Cette déclaration est implicite et les vérifications nécessaires seront faites par l'éditeur de liens.

Le schéma de transfert est placé dans le chapitre "instructions" dans la description de l'instruction CLS.

Appel au Superviseur

CALL SUPERVISEUR (CSV)

L'ensemble des sections du Superviseur et des sections constituées par les sous-programmes communs, forme le "système d'exploitation" résident.

Nous appelerons "section système" une section du système d'exploitation.

Une "section système" :

a) reste au niveau de priorité du programme appelant.

b) travaille à la fois sur les données de la tâche appelante et sur des données locales qui lui sont propres.

c) s'exécute automatiquement en mode maître.

De plus, comme c'est G qui identifie une tâche, il est logique d'associer l'appel à G et non à L.

Dans l'appel au Superviseur, c'est-à-dire à une section système, G jouera donc le rôle qu'avait L dans le CALL SECTION.

Le point c, associé aux modes d'adressage de classe 0 (voir chapitre IV), assure b puisqu'une section système peut :

- accéder à ses données propres en mode DL, IL et ILX, étant entendu que ces données ont des adresses absolues, donc que les sections système sont résidentes avec une base locale implicite nulle. (En ce sens, le système d'exploitation constitue un seul programme).

- accéder aux données de la tâche appelante en modes DG et IGX puisque la base G reste celle de la tâche appelante.

A l'exécution d'un RSV (retour au Superviseur), la machine restaure les registres L et P, avec les valeurs qui avaient été préservées, dans la zone TWB du programme appelant.

Le mode de l'appelant est restitué automatiquement par le RSV.

Le schéma de communication de l'appel au Superviseur est donné au chapitre "Instructions" dans la description de l'instruction CSV.

Exemple de programmation d'un section réentrante

Module M:MOVE du moniteur MOB destiné à déplacer une chaîne d'octets.

• Programme principal

```
PRINC   CDS
        RES     16          } TWB
        FIN
LDS1    LDS
CH1     TEXT    "ABCDEFGHIJKL"
CH2     RES,1   12
        FIN
LPS1    LPS     LDS1
DEB     LEA     CH2
        XAX                 } Chargement des paramètres
        LEA     CH1
        LDE     =12
        CSV     M:MOVE
        FIN     DEB
        END     LPS1
```

Module réentrant M:MOVE

```
FICTIV      CDS     DUM
            RES     4
            RES     1
T1          RES     1
T2          RES     1
T3          RES     1
T4          RES     1
T5          RES     1
T6          RES     1
T7          RES     1
N3          RES     1
N2          RES     1
N1          RES     1
N0          RES     1
            FIN
SUPER       LDS
            FIN
M:MOVE      LPS     SUPER       Constitue l'entrée du module par CSV
            SPA     ≠ N0
            BRU     $+2
            RSV
R:MOVE      XEX                 Constitue l'entrée du module sur branchement
            DST     ≠ T0
C           DCX     =1
            BCF     O
            LBR     @≠ T1,X
            SBR     @≠ T0,X
            BRU     C
O           LDA     ≠ T1
            BRU     @≠ N0
            FIN
            END
```

TWB fictive. Ne génère aucun binaire. Est utilisé pour générer correctement les déplacements des instructions du LPS.
La base G n'étant pas modifiée lors d'un CSV, M:MOVE travaillera dans la TWB du programme principal.

III-7. PRINCIPES DE GESTION DU SYSTEME

Fonctions du Superviseur

- Gestion des tâches : connexions aux niveaux d'interruption, mise en file d'attente, fonction "distributeur".

- Gestion des entrées/sorties : initialisation sur appel utilisateur, contrôle des fins de transfert sur interruption, etc...

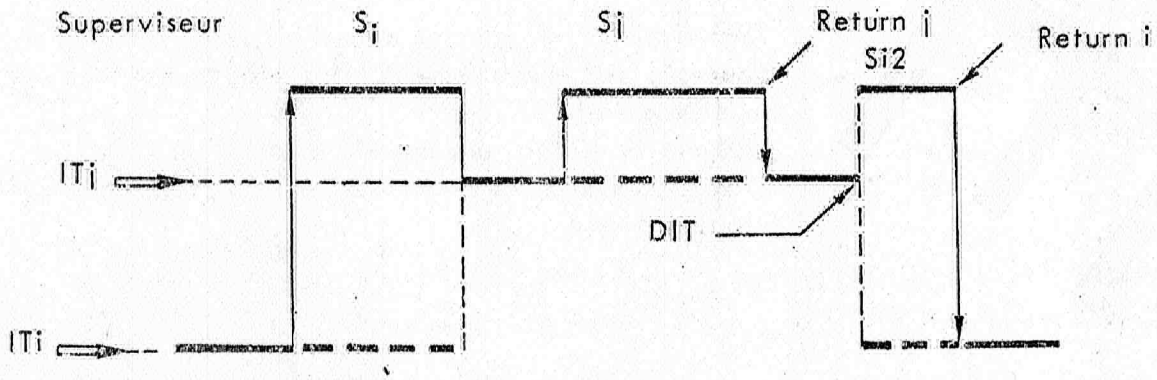
- Gestion des ressources : réservation et libération sur appel utilisateur.

- Gestion des évènements.

- Gestion de délais.

- Etc...

Fonctionnement et réentrance des appels aux superviseurs



Traitement superviseur au niveau de la tâche

Le superviseur travaille au niveau de la tâche appelante ce qui assure la protection du contexte superviseur au niveau de celle-ci.

D'autre part, pour qu'il soit tout à fait réentrant, les données variables traitées par le superviseur doivent être rangées dans une zone fournie par la tâche appelante : c'est le rôle de la zone TWB définie au paragraphe III-5.

Gestion des zones de données

• Zone des données communes au système

Cette zone est située en haut de mémoire afin qu'une adresse relative à une base G quelconque puisse toujours être positive.

Elle se compose d'un ensemble de blocs de longueur fixe qui peuvent être attribués dynamiquement sur demande des utilisateurs.
Pour assurer la translatabilité des programmes vis-à-vis de cette zone, c'est l'adresse ZC de l'ensemble de la zone commune qui sera rangée par le superviseur en G + 6, en valeur relative à G, l'adresse du bloc effectivement attribué étant fournie dans un registre (X de préférence) en valeur relative à ZC. C'est le loader qui implante cette adresse.

La tâche accédera au bloc en mode IGX avec :

- De = 6.

- (G + 6) = ZC.

- (X) = Adresse du bloc relative à ZC

la progression dans le bloc étant obtenue par incrémentation ou décrémentation de X.

L'intérêt de cette procédure est évident :

Lorsqu'un programme est translaté dynamiquement, après swapping par exemple, il suffit au système de mettre à jour le contenu de (G + 6) avec la nouvelle adresse relative ZC pour assurer la liaison avec ses données sans que l'utilisateur ait à s'en préoccuper.

Cette zone a donc pour objet principal de :
- assurer les communications entre programmes séparés
- permettre la translatabilité dynamique des programmes.

• Zones de données communes à un programme

Chaque programme possède une CDS (Common Data Section) à laquelle il accède en mode d'adressage général. Cet adressage étant toujours relatif à la base G, l'utilisateur peut ignorer l'implantation exacte de son programme, elle n'aura pas d'influence sur la programmati

• Zones de données locales

Chaque section d'un programme, pouvant disposer d'une zone de données locales, il est nécessaire de mettre à jour cette base en début de section quand on entre dans une section appelée ou lorsque l'on revient dans la section appelante.

Ce travail est exécuté automatiquement à l'insu du programmeur à qui l'on demande seulement d'indiquer le nom de la section qu'il veut exécuter.

L'éditeur de liens construit une table d'implantation relative des sections où pour chacune d'entre elles figure l'adresse relative de son point d'entrée et de son segment de données locales.

Il est possible que pour certains programmes, on désire accéder en direct à plus de 256 octets dans un segment de données. Il existe pour ce faire des instructions "incrément" et "décrément" de la base L qui permettent de déplacer la zone directement accessible. La directive Base permet d'indiquer à l'assembleur que cette zone a été déplacée.

4. Langage d'assemblage

L'assembleur est un programme de traduction de langage. Il délivre un texte objet à partir d'un texte source écrit en "langage d'Assemblage".

La tâche des programmeurs est facilitée par :

- les directives permettant d'engendrer des données sous des formes variées.
- la possibilité de répartir le travail de programmation entre plusieurs programmeurs (découpage en sections et segments).
- la possibilité d'écrire simplement des sous-programmes réentrants, étant donné la disjonction totale entre la zone de données et la zone exécutable.

L'assembleur travaille en un seul passage au cours duquel :

- chaque ligne du texte source est lue
- les symboles sont enregistrés en table.
- les allocations d'adresses sont faites en valeur relative au début des segments déclarés.
- les directives sont exécutées.
- le texte objet "binaire translatable" est édité, accompagné du dictionnaire des références satisfaites, de la liste-objet et d'une liste des erreurs qui ont pu être décelées à ce niveau.
- les références en avant "qui ne peuvent être résolues" par l'assembleur sont, en fait, traitées par l'éditeur de liens.

La configuration minimale nécessaire à l'assembleur MITRAS 1 est la suivante :

- 4 K mots de mémoire (y compris le traitement des entrées-sorties).
- une téléscriptrice avec lecteur/perforateur de ruban.

L'assembleur étendu MITRAS 2 nécessite un module mémoire supplémentaire de 4 K mots.

Nota important :

1. Dans la suite du texte, les dispositifs qui ne figurent que dans MITRAS 2 sont signalés en marge du texte par un pointillé.

   Les directives qui ne sont pas admises par MITRAS 1 sont soulignées.

2. L'assembleur MITRAS 1 nécessite environ 4 800 octets (sans table d'étiquette) ce qui autorise, sous moniteur de base MOB, environ 1 000 octets de tables, soit 100 étiquettes communes.

Les instructions du langage source sont de deux types :

- Les instructions en langage machine : elles sont traduites par l'assembleur en un seul mot-machine donnant lieu à une instruction exécutable par la logique de MITRA 15. On les appellera par la suite "instructions".
- Les instructions d'assemblage ou directives : ce sont des ordres destinés à l'assembleur, soit pour contrôler l'assemblage, soit pour générer des données ou des textes. On les appellera par la suite "directives".

IV-1. FORMAT d'une LIGNE en LANGAGE SOURCE

IV-1.1. Ligne instruction ou directive

Une ligne instruction ou directive comporte au maximum quatre zones :

- la zone étiquette :

Elle commence obligatoirement en colonne 1, contient un symbole de 1 à 6 caractères alphanumériques dont le premier est une lettre, et se termine par une colonne vide.

- la zone de commande :

Elle débute sur la première colonne non vide qui suit la zone étiquette (ou la première colonne non vide qui suit la colonne 1 si la zone étiquette n'est pas utilisée) et se termine par une colonne vide. Une commande est obligatoire pour chaque instruction ou directive.

- la zone d'argument :

Elle débute sur la première colonne non vide qui suit la zone de commande et se termine par une colonne vide, sauf si la première colonne non vide contient le caractère spécial '*' auquel cas la zone d'argument est considérée comme vide.

La zone argument ne doit pas exéder : pour MITRAS 1 la colonne 57
pour MITRAS 2 la colonne 72

- la zone de commentaire :

Elle débute sur la première colonne qui suit le caractère spécial '*'.

IV-1.2. Lignes commentaire

Une ligne commentaire est une ligne dont le 1er caractère non blanc est le signe * . L'assembleur ignore les lignes commentaires mais les reproduit sur la liste.

IV-1.3. Lignes Vierges

Les lignes vierges sont acceptées et considérées comme des cartes commentaires vides.

IV-2. CARACTERES DE BASE

L'assembleur accepte les caractères suivants :

- alphabétiques :

  A à Z ainsi que ":".

- numériques :

  0 à 9.

- spéciaux :

  Blanc + - * / . , ( ) || = # $ % & @ etc...

Tous les caractères reconnus par les organes périphériques d'entrée sont admis par l'assembleur. Ces caractères constituent un sous-ensemble du code EBCDIC.

L'assembleur ne fait aucun contrôle sur les caractères apparaissant en zone commentaire ou dans une chaîne de caractères.

IV-3. SYMBOLES

Un symbole est une suite de 1 à 6 caractères alphanumériques, le premier étant nécessairement une lettre. Il ne peut contenir ni blanc ni caractère spécial.

Un symbole est défini lorsqu'il apparaît dans la zone étiquette d'une ligne source.

Dans tous les cas, il sert à identifier la ligne source sur laquelle il apparaît.

Dans certains cas, il sert à identifier l'emplacement en mémoire du code engendré par la ligne source. Une valeur lui est alors associée : c'est l'adresse mémoire de l'octet le plus significatif.

La directive EQU permet d'affecter des valeurs numériques aux symboles.

IV-3.1. Commandes interdisant l'assignation d'une valeur à l'étiquette

Ce sont les directives :

GOTO, BASE, BND, DEF, REF, FIN, END, PAGE.

Un symbole peut apparaître en zone étiquette devant l'une de ces commandes.

Sa seule utilité est de pouvoir repérer la ligne où il apparaît (dans la zone argument d'une directive GOTO.)

IV-3.2. Commandes entrainant l'assignation d'une valeur à l'étiquette.

- Commande d'assignation de valeur :

  EQU.

- Commandes de génération :

  Instructions machine.

  Directives de génération :

  RES, DATA, GEN, TEXT, DO.

  Directives de sectionnement :

  CDS, LDS, IDS, LPS.

Tout symbole apparaissant en zone étiquette devant une commande de ce type est rangé dans la table des symboles de l'assembleur et une valeur d'adresse lui est assignée.

La valeur d'adresse est toujours relative au début du segment dans lequel le symbole apparait en zone étiquette.

Une valeur d'adresse apparaissant en zone opérande des directives DATA et GEN sera translatée par l'éditeur de liens d'une valeur égale à la base L ou P du segment où elle a été définie de façon à être relative à la base de translation G du programme.

Cette valeur pourra encore être translatée par le chargeur d'une valeur égale à la base générale, pour les programmes résidents déclarés en mode maître puisque dans ce cas les adresses indirectes en mode local doivent être absolues.

Il est cependant possible dans un segment de données LDS de forcer une expression d'étiquette à rester relative à G même si le programme est exécutable en mode maître.

Il suffit, pour cela, de faire précéder l'expression d'étiquette correspondante du caractère " ## ".

Le procédé est autorisé dans une CDS, bien qu'il soit alors ineffectif par principe.

IV-4 · CONSTANTES

Les données peuvent être introduites directement en langage d'assemblage sous forme de constantes alphanumériques. Ces constantes sont de trois types :

IV-4.1. Constante décimale entière

Une constante décimale entière est représentée par un nombre décimal entier de 5 chiffres au maximum.

Exemple :

75

La valeur absolue maximum pour une constante décimale entière de $2^{16} - 1 = 65\,535$.

La constante engendrée par l'assembleur est de la forme binaire pure, et occupe le champ défini dans la directive de génération.

IV-4.2 Constante hexadécimale

Une constante hexadécimale est représentée par un nombre hexadécimal entier de 1 à 4 chiffres hexadécimaux précédé immédiatement du caractère spécial "&".

Exemples

&1A &E3FF

IV-4.3. Chaine de caractères

Une chaine de caractères est composée de caractères alphabétiques, numériques ou spéciaux mis entre guillemets simples. La représentation interne des caractères normalisée est le code "EBCDIC" (un module inclus dans tous les moniteurs standards permet d'assurer les transcodages ASCII-EBCDIC et EBCDIC-ASCII éventuellement nécessaires). Ce transcodage est assuré automatiquement par le système d'entrée-sortie lorsque c'est nécessaire.

Exemple :

" SUITE DE CARACTERES"

Un guillemet de la chaine est représenté par deux doubles guillemets consécutifs.

"CARACT" "SUIVANT" représente la chaine CARACT"SUIVANT.

IV-5. EXPRESSIONS

Une expression se compose d'un ou plusieurs symboles ou constantes reliés entre eux par des opérateurs.

Une expression est représentée par une valeur unique calculée par l'assembleur ou l'éditeur de liens suivant des règles exposées au paragraphe IV-5.2.

Une expression est dite calculable quand sa valeur peut être déterminée lorsqu'on la rencontre pour la première fois : elle ne peut donc pas contenir de références anticipées ni de références externes.

IV-5.1. Opérateurs

L'assembleur accepte les opérateurs suivants :

- \- Moins unaire (exemple : - 3)
- \- Soustraction (exemple : A - 3)
- \+ Addition

Dans le cas où le moins unaire est suivi par une constante, celle-ci est générée par l'assembleur sous une forme binaire pur, en complément à deux.

IV-5.2. Evaluation des expressions

Nous distinguerons deux types de symboles :

- Etiquette :

Symbole qui identifie un emplacement en mémoire et dont la valeur est définie par cet emplacement .

Ce symbole peut se réduire au caractère spécial $; il désigne alors la valeur actuelle du compteur d'emplacement

- Symbole prédéfini :

Il ne définit aucun emplacement en mémoire : sa valeur est absolue et définie par des directives EQU qui précèdent son utilisation.

- Référence (anticipée) :

Symbole non encore défini. Il pourra être défini soit par une étiquette, soit par une directive EQU ou une directive REF.

- Conventions de représentation :

Les éléments de la syntaxe du langage d'assemblage seront représentés par leur dénominateur entre crochets, (ex : <expression >). Ils seront définis sous forme d'identité. La partie gauche de l'identité donne la représentation de l'élément syntaxique à définir. Le symbole d'identité sera "::=". La partie droite donne les différentes compositions de l'élément à définir. Si dans celle-ci plusieurs éléments se suivent, cela signifie qu'ils doivent figurer dans l'ordre où ils sont représentés. Par contre, s'ils sont séparés par le signe "/", cela indique qu'il faut prendre l'un OU l'autre de ces éléments.

Exemple 1 :

<ab> ::= - <valeur> / <valeur>

Dans cet exemple, "ab" peut être identique à "-valeur" OU à "valeur".

Exemple 2 :

<valeur> ::= <terme> / <valeur> <signe> <terme>

Dans cet exemple, "valeur" peut être identique à "terme" OU à "valeur", suivi de "signe", suivi de "terme". Ce qui revient à dire que "valeur" est une suite de "terme", séparés par un "signe".

Version MITRAS 1 :

<Terme> ::= <constante>/<symbole prédéfini>

<Constante> ::= <constante décimale entière>/<constante hexadécimale entière>

· Expression d'étiquette> ::= <étiquette>/<étiquette><signe><terme>

· Expression de référence> ::= <référence>/<référence><signe><terme>

· Signe> ::= +/-

· Expression prédéfinie> ::= <terme>/-<terme>/<expression d'étiquette>

Les expressions prédéfinies sont toujours calculables par l'assembleur. Les expressions peuvent n'être calculables qu'à l'édition de liens.

Version MITRAS II :

<Constante > : : = <constante décimale entière> / < constante hexadécimale entière>

<Terme > : : = <constante> / < symbole prédéfini >

<Déplacement> : : = <étiquette > - < étiquette>

<Valeur > : : = <terme > / <déplacement> / <valeur> + <déplacement> /
<valeur><signe><terme>

<Expression d'étiquette > : : = <étiquette> / <étiquette> <signe> <valeur>

<Expression de référence> : : = <référence> / < référence> <signe> <valeur>

<Expression prédéfinie > : : = <valeur> / - < valeur > / < expression d'étiquette>

<Expression> : même structure que l'"expression prédéfinie" mais toute étiquette peut être remplacée par une référence d'adresse.

Les expressions prédéfinies sont toujours calculables par l'assembleur. Les expressions peuvent n'être calculables qu'à l'édition de liens.

Remarque : Une étiquette peut se réduire au caractère spécial $ (valeur actuelle du compteur d'emplacement) mais il ne peut figurer que comme premier terme d'une expression.

Exemple : expression d'étiquette

$ + 2 autorisé

2 + $ interdit

5. Modes d'adressage

V-1. REPRESENTATION SYMBOLIQUE DES INSTRUCTIONS

V-1.1. Conventions de représentation

Dans la suite nous utiliserons les conventions de représentations suivantes :

- $\left\{\begin{matrix}=\\ \vdots\end{matrix}\right\}$ L'un des termes contenus entre les accolades peut apparaître et exclut les autres (les termes possibles sont alignés verticalement).

- $[\ ]$ Le terme entre crochets peut ne pas apparaître soit parce qu'il n'est pas obligatoire, soit parce qu'il est implicite.

- ... Répétition possible de l'expression comprise entre le séparateur de fermeture qui précède immédiatement les points de suspension et le séparateur d'ouverture qui lui est associé.

Exemples :

$$\left\{\begin{matrix}A\\ B\\ C\end{matrix}\right\} \quad [,D]$$

L'un des termes A, B ou C doit être présent; le terme ,D est optionnel.

$$\left\{\begin{matrix}[A]\\ B\\ C\end{matrix}\right\}$$

L'un des termes A, B ou C doit être utilisé; si c'est le terme A : il peut ne pas être indiqué.

$$\left\{\begin{matrix}A\\ D\end{matrix}\right. \quad [,B] \ \ldots, C \left.\right\} \ \ldots$$

L'expression entre accolades peut être répétée; dans le premier terme possible, l'élément de type B peut être répété.

V-1.2. Représentation des instructions

L'écriture des instructions obéit aux conventions de format suivantes :

$$[\text{Etiquette}] \quad \text{OP} \quad \left[\left\{\begin{matrix}\left\{\begin{matrix}=\\ @\end{matrix}\right\} & D & [,X]\\ [*] & D & \\ @* & D & ,X\end{matrix}\right\}\right]$$

OP : Code opération

D : Déplacement

= : La valeur de l'opérande est égale au déplacement (paramètre)

@ : Indirection

: Adressage relatif à la base générale (CDS)

,X : Indexation

Remarque : Pour les noms d'instruction ou de directive comportant quatre caractères, seuls les trois premiers sont utilisés pour les reconnaître.

V-2. REPRESENTATION DE L'ADRESSAGE

V-2.1. Types d'adressage

Les possibilités d'adressage du MITRA 15 sont adaptées en fonction des codes opérations des diverses instructions.

On distingue trois groupes de fonctions d'adressage qui correspondent à trois types d'instructions :

- Type 0

Ce sont les instructions de :

- chargement et rangement des registres.
- opérations arithmétiques fixe ou flottante.
- opérations logiques.
- traitement de chaine de caractères.
- comparaison.

- Type 2

Ce sont les instructions de branchement conditionnel ou inconditionnel.

- Type 1

Ce sont les instructions de :

- décalages.
- opérations sur index.
- opérations sur bases.
- appels section ou superviseur.
- entrées-sorties.
- opérations sur registre.
- opérations sur interruptions et masque d'interruptions.

L'ensemble de ces trois catégories forme un jeu extrèmement complet qui sera examiné par ailleurs après avoir étudié les formes d'adressage propres à chacun des types.

Les adressages sont représentés en fonction des conventions suivantes :

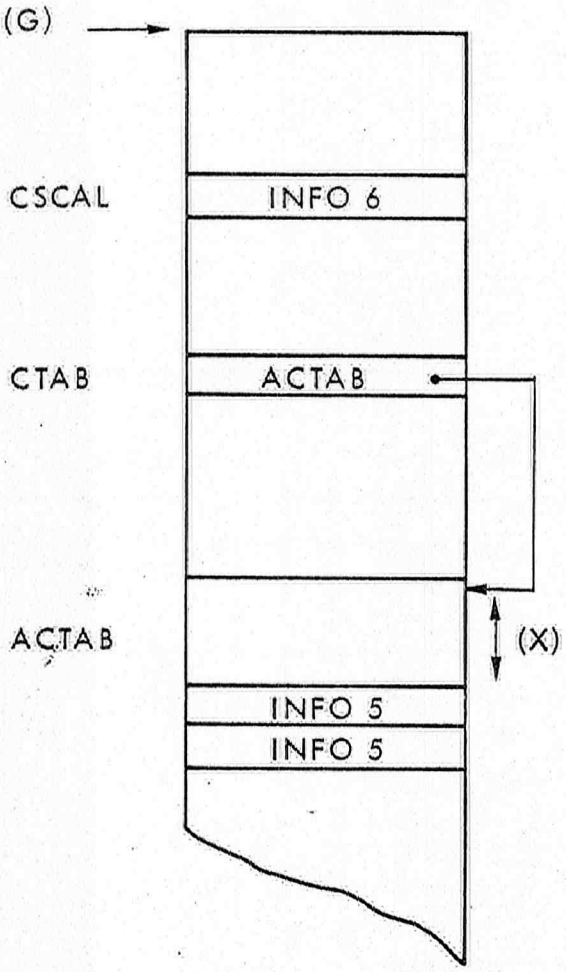
- L Base locale
- G Base générale
- G' Base générale en mode esclave et zéro en mode maître
- X Registre d'index
- P Base programme
- D Déplacement
- ( ) Contenu de

■ Adressage de type 0

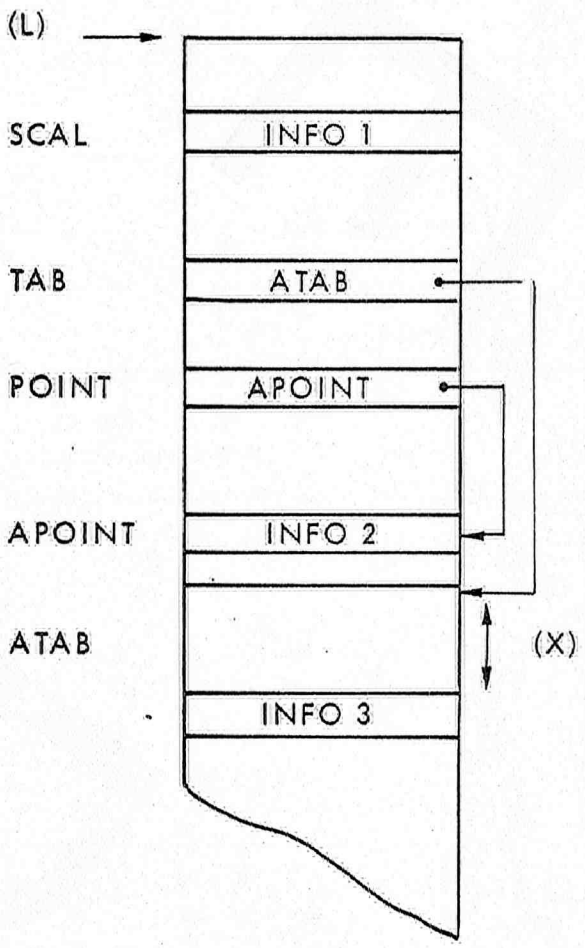
Mode	Langage d'assemblage	Donnée adressée	Fonction d'adressage
Direct local DL	IDENT	Octet, mot ou double mot situé dans les 256 premiers octets du segment local.	Y=(L)+D
Indirect local IL	@ IDENT	Octet, mot ou double mot situé à un emplacement quelconque et pointé à travers le segment local.	Y=G'+((L)+D)
Indirect local indexé ILX	@ IDENT, X	Elément d'un tableau d'octets, de mots ou de double-mots situé à un emplacement quelconque et pointé à travers le segment local.	Y=G'+((L)+D)+(X)
Direct général DG	*IDENT	Octet, mot ou double-mot situé dans les 256 premiers octets du segment commun.	Y=(G)+D
Indirect Général Indexé IGX	@*IDENT, X	Element d'un tableau pointé à travers le segment commun.	Y=(G)+((G)+D)+(X)
Paramètre P	=OPERAND	L'opérande d'un octet figure dans l'instruction. Cet octet est éventuellement étendus par 8 zéros en poids forts.	Y = D

Exemples d'adressage type 0

Segment commun



Segment local



Instruction		Opérande
LDA	#CSCAL	INFO 6
DLD	@ #CTAB, X	INFO 5

Instruction		Opérande
LDA	SCAL	INFO 1
LDA	@POINT	INFO 2
LBR	@TAB, X	INFO 3

Instruction		Opérande
LDA	= INFO 4	INFO 4

■ Adressage de type 1

Les instructions de ce type sont :

- soit des instructions sans opérande : échange de registres, fin de section, etc...

- soit des instructions dont l'opérande est le plus souvent connu (ou connu à un modificateur près) lors de l'écriture du programme : décalage, incrément, index, etc...

On distingue quatre modes d'adressage :

Mode	Ecriture en Assembleur	Opérande	Fonction
Paramètre P	= PARAM	L'opérande est défini par la valeur du déplacement.	Y = D
Paramètre indexé PX	= PARAM, X	L'opérande est défini par la valeur du déplacement augmenté du contenu de X.	Y = D + (X)
Direct local DL	IDENT	L'opérande est placé dans les 256 premiers octet du segment local.	Y = (L) + D

Remarque 1 :

Pour simplifier l'écriture des programmes, certains codes d'instruction symboliques reconnus par l'assembleur spécifient le code opération et le déplacement.

Par exemple l'instruction sur registres SRG est spécifiée par le déplacement.

```
SRG     = 02      échange de A et E
SRG     = 04                  A et X
SRG     = 06                  E et X
 :
SRG     = 1C        -A →A
 :
etc
```

Pratiquement l'assembleur reconnaît

```
XAE                 équivalent de  SRG = 02
XAX                                SRG = 04
XEX                                SRG = 06
 :                                  :
CNA                                SRG = 1C
 :
etc
```

De plus en ce qui concerne les décalages l'assembleur MITRAS 2 reconnaît 14 mnémoniques qui sont des spécifications des deux codes opérations SHR et SHC.

Exemples :

SHR = &23 est équivalent de SRCS = 3 (décalage droit circulaire simple)

SHR = &E8 est équivalent de SRCD = 8 (décalage droit circulaire double)

SHC = &0B est équivalent de SLLD = 11 (décalage gauche logique double)

SHC = &4E est équivalent de SRLD = 14 (décalage droit logique double)

Remarque 2 :

Cas des instructions CLS et CSV

Il y a deux modes d'emploi des instructions CLS et CSV.

a) L'opérande est un nom de LPS. L'assembleur génère un mot nul. C'est l'éditeur de liens qui d'une part détermine s'il doit générer une instruction CLS ou CSV suivant le type de la section (section moniteur ou section utilisateur) et d'autre part détermine le numéro de section correspondant.

C'est l'utilisation normale, l'utilisateur n'ayant pas à se soucier du numéro de section.

b) L'opérande n'est pas un nom de LPS.

Ces instructions sont traitées comme des instructions classe 1 quelconques, les trois modes d'adressage étant utilisables.

Exemple pour un module de programme

```
PROG    CDS
        RES     16
        FIN
L1      LDS
        FIN
P1      LPS     L1
        RTS
        FIN     P1
L2      LDS
        FIN
P2      LPS     L2
        RTS
        FIN     P2
L3      LDS
        FIN
P3      LPS     L3
        RTS
        FIN     P3
```

```
L4          LDS
            RES     3
NUMSEC      DATA    3
            FIN
P4          LPS     L4
DEB         CLS     S1      ---> Appel de S1
            CLS     = 2     ---> Appel de S2
            LDX     = 1
            CLS     = 2, X  ---> Appel de S3
            CLS     NUMSEC  ---> Appel de S3
            CSV     M:EXIT  ---> Appel moniteur
            FIN     DEB
            END     P4
```

Ces utilisations nécessitent la connaissance du numéro de section dans la PRT du programme.

Note : La possibilité CLS NUMSEC n'est pas fournie dans MITRAS 1.

■ Adressage de type 2

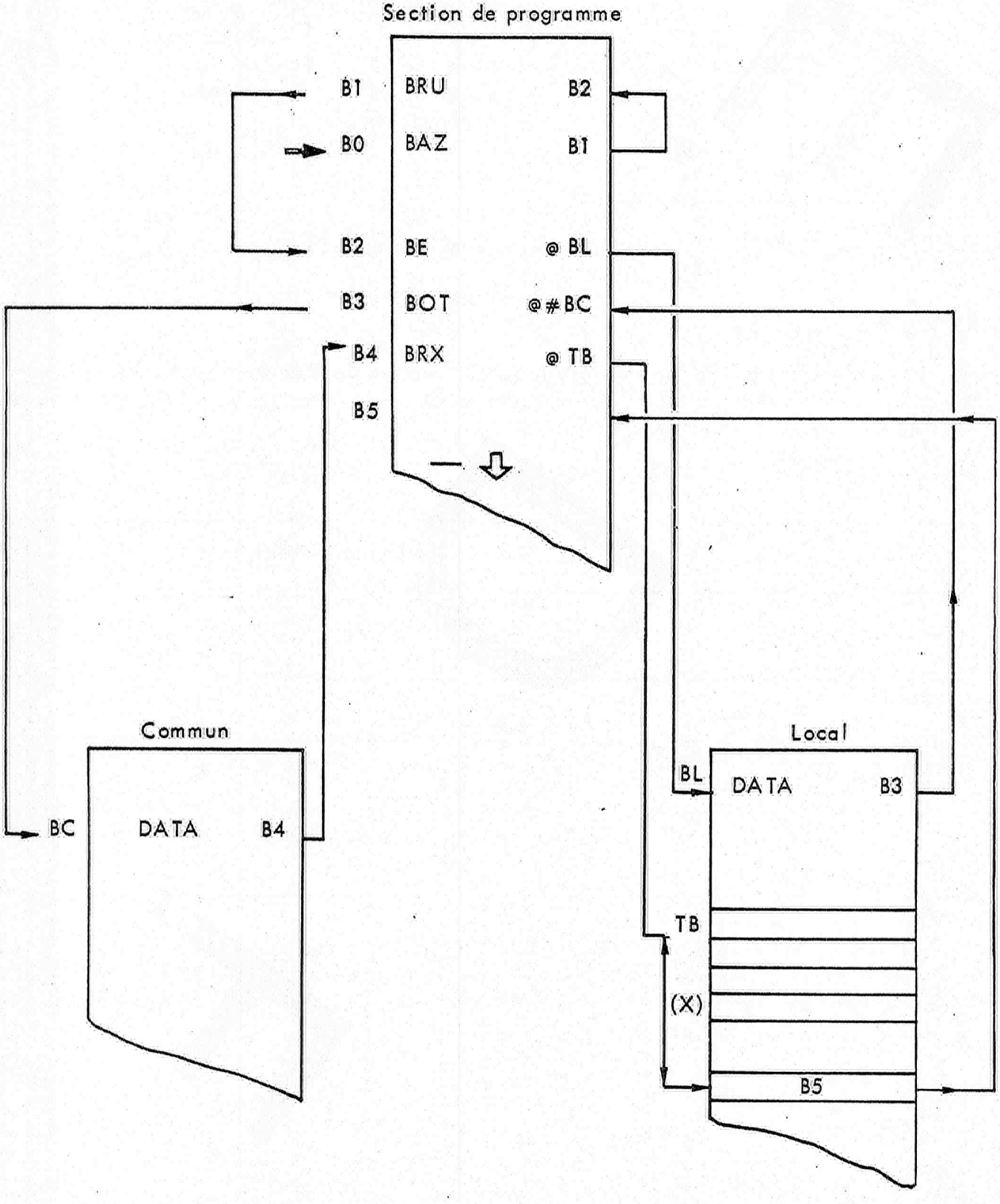
Les instructions désignées par une instruction de rupture de séquence appartiennent normalement à la même section de programme que cette rupture. Les sections de programme peuvent cependant avoir une longueur quelconque.

On distingue quatre modes d'adressage :

Mode	Ecriture en Assembleur	Instructions de branchement	Fonction
Relatif plus RP	LABEL	Toute instruction à moins de 512 octets en aval.	Y=(P)+2 D
Relatif moins RM	LABEL	Toute instruction à moins de 512 octets en amont.	Y=(P)-2 D
Indirect local IL	@LABEL	Toute instruction pointée à travers le segment local.	Y=G'+((L)+ D)
Indirect général IG	@*LABEL	Toute instruction pointée à travers le segment commun.	Y=G'+((G)+D)

Par ailleurs il existe une instruction de branchement inconditionnel indexé. Si le branchement est indirect l'indexation est une pré-indexation (préférable pour traiter les Tables de branchements) contrairement à l'indexation des données qui est une post-indexation (préférable pour accéder des éléments de tableau).

Exemples :


Section de programme
B1
BRU
B2
B0
BAZ
B1
B2
BE
@ BL
B3
BOT
@#BC
B4
BRX
@ TB
B5
Commun
BC
DATA
B4
Local
BL
DATA
B3
TB
(X)
B5

V-2.2. Expressions autorisées

- Classe 0

Expression prédéfinie ou de référence

- Classe 1

P : valeur

PX : valeur

DL : expression prédéfinie ou de référence

- Classe 2

RP : expression de référence

RM : expression d'étiquette

DL : expression prédéfinie ou de référence

DG : expression prédefinie ou de référence

6. Directives

VI-1. SECTIONNEMENT DU TEXTE SOURCE

VI-1.1. Généralités

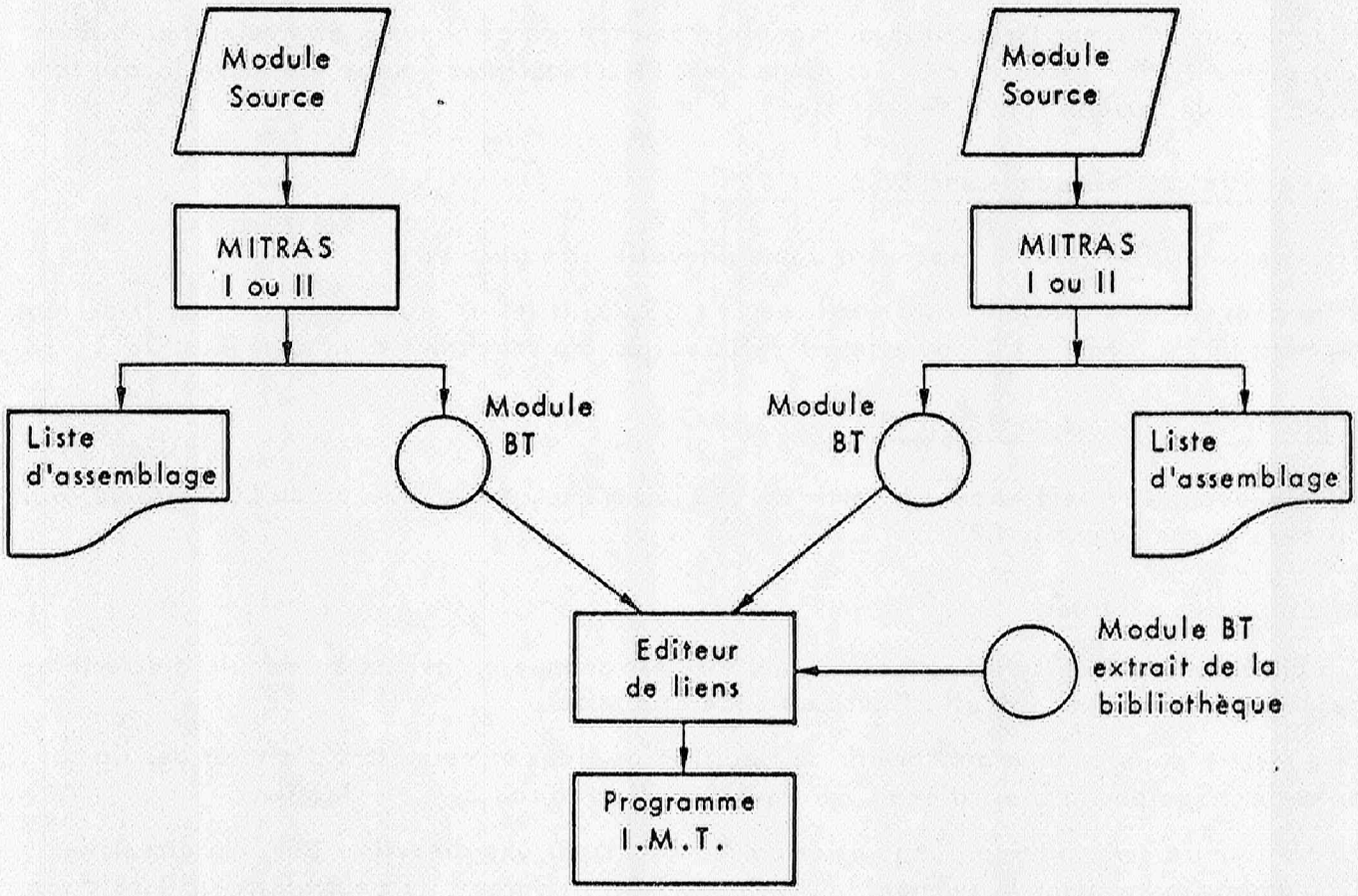
L'assembleur transforme le texte source en un module objet en format binaire translatable (BT). Il ne peut satisfaire que les références à des symboles du texte source assemblé.

C'est l'éditeur de liens qui transforme les modules assemblés en un programme complet représenté par une image mémoire translatable (IMT). Toutes les références externes (noms de sections) entre modules du programme sont alors satisfaites.

Le chargeur charge l'IMT en mémoire à partir d'une base générale G qui n'est définie qu'à cette phase de chargement.

VI-1.2. Texte source

Le texte source constitue l'unité d'assemblage. Il comporte un ou plusieurs segments et se termine obligatoirement par une ligne comportant la directive END.



VI-1.3. Section commune

La section de données communes (CDS) effective ou fictive doit toujours, si elle existe, être déclarée avant tout segment du module d'assemblage.

VI-1.4. Sections

Chaque section comporte nécessairement un segment de programme exécutable (LPS) qui définit la section. Un segment de données locales (LDS) est normalement associé à un LPS.

S'il est défini effectivement dans le même module, ce LDS précède obligatoirement le LPS dans le texte source. Plusieurs segments LPS peuvent être associés au même LDS. Il y a autant de sections que de LPS.

VI-1.5. Portée des identificateurs

On distingue les étiquettes définies dans le module d'assemblage et les étiquettes externes déclarées à l'aide des directives DEF et REF.

■ Etiquettes internes

• Etiquettes définies dans la CDS

Elles sont définies sur l'ensemble du module d'assemblage et peuvent être référencées dans tout segment. En conséquence elles ne peuvent être redéfinies comme étiquette locale sans provoquer une erreur "double-définition".

• Etiquettes définies dans un LDS

Elles sont définies jusqu'à rencontre d'une nouvelle directive LDS.

Elles peuvent être référencées à partir de la CDS, dans le LDS lui-même et à partir de tout segment LPS suivant le LDS ou elle est définie, jusqu'à rencontre d'un nouveau LDS.

• Etiquettes définies dans un LPS

Elles peuvent être référencées à partir du LPS lui-même, de la CDS ou du LDS associé (qui précède normalement le LPS).

■ Etiquettes externes

On dit qu'une étiquette est externe quand elle est connue en dehors du module d'assemblage où elle est définie (où elle figure en zone étiquette).

Elle restera donc connue au moment de l'édition de liens et permettra d'assurer des liaisons autrement que par la biais d'une CDS (réelle ou fictive) ou de Call Section.

Une étiquette sera externe quand elle sera déclarée par une directive DEF. La directive DEF devra figurer dans le segment où l'étiquette est définie. Elle pourra être utilisée dans un autre module d'assemblage à condition qu'elle y soit déclarée par une directive REF. La directive REF figurera dans le segment où l'étiquette externe est utilisée.

Le statut d'étiquette externe ne modifie pas pour l'assembleur les notions d'étiquette commune ou locale.

Quand on désire déclarer dans une directive REF une étiquette appartenant à la CDS, on la fera précéder du caractère * .

Exemple :

```
REF   *ETIQU1, ETIQU2, *ETIQU3
```

Il ne faut pas confondre étiquettes externes et nom de segment. Les noms de segment bien que connus en dehors du module d'assemblage au moment de l'édition de liens ne sont accessibles que par Call Section ou Call Superviseur.

Exemple :

```
PROG      CDS
          DEF       ETIQ0
          RES       16
ETIQ0     DATA      1
          FIN

LDS1      LDS
          RES       8
          DEF       ETIQ1
          REF       ETIQ3
ETIQ1     DATA      2
          DATA      ETIQ3
          FIN

LPS1      LPS       LDS1
          REF       ETIQ2
Z         LDA       ETIQ2
          FIN       Z
          END

LDS2      LDS
          REF       ETIQ1
          REF       *ETIQ0
X         DATA      ETIQ1
ETIQ2     DATA      0
ETIQ3     DATA      4
          FIN
```

```
LPS2        LPS         LDS2
Y           LDA         #ETIQ0
            STA           X
            LDA         ETIQ1
            FIN         X
```

Remarque :

Il faut bien prendre garde pour les étiquettes externes locales que le déplacement est généré relativement au LDS de la section où elles figurent en DEF mais est utilisé relativement au LDS de la section où elles figurent en REF.

VI-1.6. Compteur d'emplacement

Le compteur d'emplacement est toujours exprimé en adresse octet et peut atteindre une valeur maximum de $2^{16} - 1 = 65\,535$.

Le compteur d'emplacement est représenté symboliquement par le caractère spécial $.

Ce compteur est remis à zéro à chaque déclaration de segment, de sorte que les références calculées à l'assemblage sont toujours relatives au début du segment déclaré.

VI-1.7. Directives de sectionnement

Les directives de sectionnement permettent de définir la structure du module d'assemblage en sections et segments.

Ce sont :

- Section commune : CDS
- Segment de données locales : LDS
- Segment indirect de données : IDS
- Segment de programme exécutable : LPS
- Fin de segment : FIN
- Fin de module : END

Chaque segment ouvert par une directive de sectionnement doit se terminer par une directive FIN.

■ Directive CDS/FIN

La directive CDS identifie la section de données communes CDS.

Format :

Etiquette	Commande	Argument
<Nom>	CDS ⋮ FIN	[DUM]
[<étiquette>]		

Résultat :

- Le compteur d'emplacement est remis à zéro.
- Toutes les étiquettes pourront être référencées dans les sections déclarées dans le module.
- Si l'option DUM est présente, aucun code n'est généré, la section est fictive.

■ Directive LDS/FIN

La directive LDS identifie un segment de données locales LDS.

Format :

Etiquette	Commande	Argument
<Nom>	LDS ⋮ FIN	[DUM]
[<étiquette>]		

Résultat :

- Le compteur d'emplacement est remis à zéro.
- Le nom défini en zone étiquette constitue une définition externe implicite.
- Si l'option DUM est présente, aucun code n'est généré.

Directive IDS/FIN

La directive IDS identifie un segment de données à accès indirect, à l'intérieur d'un LDS ou d'une CDS, tel que toute étiquette apparaissant entre la directive IDS et la directive FIN associée soit définie en valeur relative au début du segment indirect déclaré.
Cette valeur relative a la nature d'un nombre et non plus d'une adresse.

Format :

Etiquette	Commande	Argument
<Nom>	IDS	[DUM]
[<étiquette>]	FIN	

Résultat :

- La valeur du compteur d'emplacement est préservée. Le compteur d'emplacement est remis à zéro.

Après la fermeture du segment IDS par la directive FIN, la valeur initiale du compteur d'emplacement est restituée, augmentée ou non de sa valeur relative actuelle suivant que l'IDS est effectif ou fictif.

- Si l'option DUM est utilisée, aucun code n'est généré, le segment est fictif.

- L'étiquette définie par la directive IDS est considérée comme une étiquette normale du LDS si le segment IDS est effectif, elle prend une valeur nulle si le segment IDS est fictif.

Directive LPS/FIN

La directive LPS identifie un segment de programme exécutable LPS et, par là, une section de programme.

Format :

Etiquette	Commande	Argument
<Nom 1>	LPS	<Nom 2>
[<étiquette>]	FIN	<Etiquette 2>

Résultat :

- Le compteur d'emplacement est remis à zéro.

- <Nom 1> constitue le nom de LPS et de section; c'est une définition externe implicite. Le nom référencé dans un CALL SECTION constituera la référence externe implicite correspondante.

- <Etiquette 2> définit l'adresse effective de début d'exécution de la section.

- <Nom 2> référence le LDS associé.

■ Directive END

La directive END indique la fin du module d'assemblage.

Format :

Etiquette	Commande	Argument
[<Etiquette>]	END	[<Nom de section>]

Résultat :

- L'assemblage du module est terminé.

- Le nom de section indiqué dans la zone argument définit la section à lancer après chargement du programme.

Cette étiquette constitue une référence externe implicite.

- Au cours de l'édition de liens, il ne doit apparaitre qu'une seule directive END comportant en zone argument une déclaration de nom de section.

■ Traitement du %EOD

Il est à noter que l'assembleur a comme entrée symbolique un fichier source se terminant donc normalement par un enregistrement fin de fichier (%EOD).

Cette fin de fichier n'est pas strictement nécessaire à l'assembleur pour reconnaître la fin du programme.

Cependant l'absence d'un %EOD rendra aléatoire la suite des opérations, et ce plus particulièrement sous module d'enchaînement.

La rencontre d'un %EOD avant une directive END (donc en cours de programme) sera signalée par une erreur de Niveau 3 et l'assemblage sera abandonné.

VI-2. DIRECTIVES D'ASSEMBLAGE

On peut distinguer les directives utilisables pour l'assemblage d'une section de données communes ou locales et les directives utilisables pour l'assemblage d'un segment de programme exécutable.

Les directives d'assemblage se répartissent de la façon suivante :

Directive	Segment de données	Segment de programme
1. Adressage :		
RES	X	X
BND	X	
BASE		X
2. Définition de symboles :		
EQU	X	X
3. Controle d'assemblage :		
GOTO	X	X
DO	X	
PAGE		X
4. Génération de données :		
DATA	X	
TEXT	X	
GEN	X	X
5. Identification de définitions externes :		
DEF	X	X
REF	X	X

VI-2.1. Assemblage d'un segment de données

■ Directives relatives à l'adressage

• Directive RES

Réservation d'une zone de mémoire.

Format :

Etiquette	Commande	Argument
[<Etiquette>]	RES [,1]	<valeur>
TOTO	RES RES,1	3 5

Résultat :

- Si l'option 1 est utilisée dans la zone commande l'unité d'emplacement est l'octet sinon c'est le mot.

- Le compteur d'emplacement est d'abord progressé à une frontière de mot; si l'unité d'emplacement est le mot.

- La valeur affectée au symbole défini dans la zone étiquette est la première adresse de la zone réservée.

• Directive BND

Avancement du compteur d'emplacement à une limite de mot.

Format :

Etiquette	Commande	Argument
[<Etiquette>]	BND	
TOTO	BND	

Résultat :

Le compteur d'emplacement est progressé jusqu'à atteindre une valeur paire, c'est-à-dire une limite de mot.

■ Directives relatives au traitement des symboles : EQU

Format :

Etiquette	Commande	Argument
<Nom>	EQU	<expression prédéfinie> /"< caractère>" / "< caractère> < caractère> "
ZON	RES	4
TOTO	EQU	ZON
TATA	EQU	ZON+2
TUTU	EQU	5
TITI	EQU	"AB"

Résultat :

- L'expression qui apparait dans la zone argument définit le symbole déclaré en zone étiquette. Cette expression ne doit pas contenir de référence en avant.
- La zone argument peut contenir un ou deux caractères alphanumériques entre guillemets.
- Le symbole $ qui représente la valeur du compteur d'emplacement peut apparaître en zone argument.
- Un symbole apparu en zone étiquette d'une directive EQU ne peut plus être redéfini.
- La chaîne n'est pas autorisée en MITRAS 1.

■ Directives de contrôle d'assemblage

• Directive GOTO

Branchement conditionnel d'assemblage.

Format :

Etiquette	Commande	Argument
<Etiquette>	GOTO ,k	<étiquette 1> , <étiquette 2> , <étiquette n>
TOTO NN TATA	GOTO ,2 EQU 2 GOTO, NN	BR1, BR2, BR3, . . . , BRn BR1, BR2, BR3

Résultat :

- L'assembleur reprend l'assemblage à la ligne qui contient en zone étiquette la k^{ième} étiquette déclarée dans la zone argument de la directive GOTO.

- K est une expression du type 'valeur' (voir pages 4-6 et 4-7) donc calculable au moment où la directive GOTO est rencontrée.

- Les étiquettes déclarées en zone argument doivent se référer à des lignes qui suivent la directive GOTO.

- Si K n'est pas compris entre 1 et n, ou s'il n'est pas calculable, un message d'erreur est édité et l'assemblage reprend à la ligne qui suit la directive GOTO.

- Si la directive END apparaît avant l'étiquette cherchée, l'assemblage est interrompu et un message d'erreur est produit.

Directive DO

Itération d'assemblage d'une instruction.

Format :

Etiquette	Commande	Argument
[<Etiquette>]	DO	<valeur>
TOTO	DO	7
	DATA	&78A2

Résultat :

- L'expression absolue déclarée en zone argument doit être calculable. Elle fournit un nombre entier $\leqslant 128$ qui représente le nombre de fois que la ligne suivante doit être assemblée.

- L'indice d'itération est représenté symboliquement par le caractère spécial '%' et peut être utilisé dans la ligne à assembler. A chaque itération, il prend la valeur du compteur d'itération.

- Le compteur d'itération commence à la valeur 1 lors de la première boucle.

- Lorsque l'itération est terminée, l'assemblage reprend normalement à la deuxième ligne qui suit la directive DO.

- Si l'expression absolue est négative ou nulle, ou si elle n'est pas calculable (auquel cas un message d'erreur est produit), l'assemblage reprend directement à la deuxième ligne qui suit la directive DO.

- Si une étiquette apparaît en zone étiquette, elle est associée au premier octet généré.

Directives de génération de données et de texte

Directives DATA

Génération de données.

Format :

Etiquette	Commande	Argument
[<Etiquette>]	DATA [,1]	[#]<exp 1> [,[#]<exp 2>] ...
TOTO	DATA DATA,1	ETIQ1,#ETIQ2,ETIQ1-ETIQ2,3,&8AF2 7,&8E, 5+4

Résultat :

- La directive DATA engendre des données dont les valeurs sont celles des expressions i.

- Une expression i est une "expression" au sens du paragraphe 4-5.2. ou une chaîne de 1 ou 2 caractères. (La chaîne n'est pas autorisée en MITRAS 1).

Les données sont engendrées chacune cadrée à droite sur un ou deux octets suivant que l'option '1' est ou n'est pas utilisée dans la zone commande.

- Si une étiquette est présente en zone étiquette, elle est associée au premier octet généré.

- Si <exp 1> est précédé du symbole #, c'est une expression qui doit rester relative à G lors d'un chargement en mode maître.

Remarque :

Pour MITRAS 2, les étiquettes opérationnelles (standard et utilisateur) sont des symboles communs implicites. Il suffit donc, pour spécifier une étiquette opérationnelle dans un bloc de commande d'entrée/sortie, d'écrire par exemple :

```
DATA,1      M:BI
```

pour spécifier l'étiquette opérationnelle M:BI.

Pour MITRAS 1, il est nécessaire soit d'indiquer directement le numéro de l'étiquette opérationnelle utilisée, soit de le définir tout d'abord par une directive EQU.

Pour tout renseignement complémentaire sur les étiquettes opérationnelles, se reporter au chapitre entrée/sortie.

Exemple :

```
MITRAS 2
    CDS
     :
     :
    FIN
```

```
LDS1    LDS
CB      DATA      0
        DATA,1    &80
        DATA,1    M:EO
        DATA      #CHAINE
        DATA      8
CHAINE  TEXT      "ECRITURE"
        FIN

LPS1    LPS       LDS1
DEB     LEA       CB
        CSV       M:IO
        LEA       CB
        CSV       M:WAIT
        CSV       M:EXIT
        FIN       DEB

        END       LPS1
```

FIN DE FICHIER

MITRAS 1

```
        CDS
         :
        FIN
```

```
LDS1    LDS
CB      DATA      0
        DATA,1    &80
        DATA,1    6
        DATA      # CHAINE
        DATA      8
CHAINE  TEXT      "ECRITURE"
        FIN

LPS1    LPS       LDS1
DEB     LEA       CB
        CSV       M:IO
        LEA       CB
        CSV       M:WAIT
        CSV       M:EXIT
        FIN       DEB

        END       LPS1
```

FIN DE FICHIER

```
        CDS
M:EO    EQU       6
        FIN
```

```
LDS1    LDS
CB      DATA      0
        DATA,1    &80
        DATA,1    M:EO
        DATA      # CHAINE
        DATA      8
CHAINE  TEXT      "ECRITURE"
        FIN

LPS1    LPS
DEB     LEA       CB
        CSV       M:IO
        LEA       CB
        CSV       M:WAIT
        CSV       M:EXIT
        FIN       DEB

        END       LPS1
```

FIN DE FICHIER

- Directive GEN

Génération de valeur

Format :

Etiquette	Commande	Argument
[<Etiquette>]	GEN, liste de zones	Liste d'expressions
TOTO	GEN,4,2,2 GEN,1,1,3,8,2,1	7,1,1 0,1,2,&F2,3,1

La directive GEN génère un octet ou un mot en fonction de la configuration spécifiée (en élément binaire).

- Liste de zones : est une liste d'expressions du type 'terme', chacune définissant la taille (en éléments binaires) de la zone générée (la somme des zones doit être égales à 8 ou 16 éléments binaires). Les zones de taille nulle sont interdites).

- Liste d'expressions : est une liste d'expressions du même type que celles définies pour la directive DATA et définissant le contenu de chaque zone générée. La valeur représentée par la liste d'expressions est mise à l'assemblage dans la zone correspondante.

Les expressions de la liste de zones et celles de la liste d'expressions doivent être séparées par des virgules.

Il y a correspondance un à un et de gauche à droite, entre les éléments de la liste de zone et ceux de la liste d'expression; le code est généré de façon que la première zone contienne la première valeur, etc...

Les valeurs sont cadrées à droite dans leur zone. La première zone correspond avec poids fort de l'ensemble.

- Directive TEXT

Génération d'une chaîne de caractères.

Format :

Etiquette	Commande	Argument
[<Etiquette>]	TEXT	"<chaîne de caractères>"
TOTO	TEXT	"CHAINE DE CARACTERES"

Résultat :

- La chaîne de caractères est assemblée en format EBCDIC dans une zone qui commence à l'adresse donnée par la valeur actuelle du compteur d'emplacement et se termine avec le dernier caractère engendré à une adresse quelconque.

- Le premier octet contient le premier caractère de la chaîne et ainsi de suite.

- Si une étiquette est présente dans la zone étiquette elle identifie le premier octet engendré.

• Directive DEF

Format :

Etiquette	Commande	Argument
[<Etiquette 1>]	DEF	<Etiquette 2>..
	DEF	ETIQ

Résultat :

DEF déclare les labels comme définitions externes.

Les labels sont définis dans la section en cours.

Un label doit être déclaré comme définition externe avant son emploi dans la section.

• Directive REF

Format :

Etiquette	Commande	Argument
[<Etiquette>]	REF	[#]<Etiquette>..
ETIQ 1	REF REF	ETIQ1 #ETIQ2

Résultat :

REF déclare les étiquettes comme des références externes.

Les labels doivent être définis avant leur usage dans la section courante, et ils seront définis au cours d'assemblage ultérieur.

Si l'étiquette fait partie de la CDS, elle doit être précédé du symbole # .

VI-2.2. Assemblage d'un segment de programme

■ Directives relatives à l'adressage

- Directives RES

Réservation d'une zone de mémoire. Voir paragraphe 6-2.1. page 6-9.

- Directive BASE

Contrôle d'adressage relatif.

Format :

Etiquette	Commande	Argument
[<Etiquette>]	BASE	[<Etiquette>]
ETIQ ETIQ1	BASE BASE	 ETIQ2

Résultat :

- L'étiquette déclarée en zone argument est une étiquette du segment de données locales.

- Toutes les adresses de LDS référencées dans le segment de programme entre deux directives BASE, sont générées en valeur relative à l'adresse définie en zone argument.

- Une directive BASE sans déclaration d'étiquette dans la zone argument, ferme simplement l'adressage relatif ouvert par la directive BASE précédente.

Pour ouvrir un autre adressage relatif, il faudra déclarer une autre directive BASE avec déclaration d'étiquette en zone argument.

- Une directive BASE avec déclaration d'étiquette en zone argument ferme l'adressage relatif de la directive BASE précédente et ouvre un adressage relatif sur la nouvelle étiquette déclarée en zone argument.

- Une directive BASE avec déclaration d'étiquette est fermée soit par une nouvelle directive BASE soit par la directive FIN qui termine l'assemblage du segment de programme.

■ Directive relative au traitement des symboles EQU

Voir paragraphe VI-2.1. page VI-9.

• Directive COMMON

Format :

Etiquette	Commande	Argument
[<Etiquette 1>]	COMMON	[<Etiquette 2 >]
ETIQ1	COMMON	ETIQ2
ETIQ3	COMMON	

Résultat :

Déclaration d'un pointeur portant sur le commun FORTRAN dont le nom se trouve en zone argument.

Lorsque l'étiquette se trouvant en zone argument est absente, le commun FORTRAN est un commun blanc.

Remarque :

L'utilisation de la directive COMMON n'est autorisé que dans le cas d'un sous-programme assembleur destiné à être utilisé avec un programme principal FORTRAN. Dans le cas contraire, une erreur est signalée par l'Editeur de liens.

■ Directive de contrôle d'assemblage

• Directive GOTO

Branchement conditionnel d'assemblage. Voir paragraphe VI-2.1. page VI-10.

■ Directive de génération d'instruction

• Directive GEN

Génération de valeur. Voir paragraphe VI-2.1. page VI-14.

L'utilisation de cette directive en zone de programme exécutable doit permettre essentiellement de générer des instructions non standard propres à une configuration spéciale.

Si après GEN, la valeur du compteur d'emplacement progressé n'est pas sur une limite de mot (utilisation d'un nombre impair de GEN d'octets), l'assembleur signale l'erreur, génère un octet nul et fait progresser d'une unité le compteur d'emplacement.

■ Directives d'identification et définition externes

• Directive DEF

Voir paragraphe VI-2.1. page VI-15

• Directive REF

Voir paragraphe VI-2.1. page VI-15

VI-3. DIRECTIVE DE SAUT DE PAGE

La directive PAGE fait imprimer la liste d'assemblage sur la page suivante, si l'organe de sortie est une imprimante.

Format :

Etiquette	Commande	Argument
[<Etiquette>]	PAGE	
ETIQ	PAGE	

7. Instructions

VII-1. GENERALITES

Ce chapitre décrit toutes les instructions MITRA 15. Elles sont regroupées en 11 familles :

- Chargements et rangements
- Arithmétique virgule fixe
- Opérations logiques
- Incrémentation de registres
- Décalages
- Opérations entre registres
- Arithmétique virgule flottante
- Traitement des chaînes
- Branchements
- Branchements systèmes
- Instructions de commande

Par ailleurs, les instructions possèdent trois caractéristiques fondamentales qui seront précisées pour chacune d'elles et qui sont :

• La classe : indiquant les modes d'adressages permis et par là le sens exact dans chaque cas de l'adresse calculée.

Classe	Modes d'adressage permis	
0	DL	Direct local
	P	Paramètre
	DG	Direct général
	IL	Indirect local
	IGX	Indirect général indexé
	ILX	Indirect local indexé
0'	DL	Direct local
	DG	Direct général
	IL	Indirect local
	IGX	Indirect général indexé
	ILX	Indirect local indexé
1	DL	Direct local
	PX	Paramètre indexé
	P	Paramètre

Classe	Modes d'adressage permis	
2	RP	Relatif plus
	RM	Relatif moins
	IL	Indirect local
	IG	Indirect général

Le fonctionnement des modes d'adressage est indiqué dans le chapitre "modes d'adressage".

Cependant certaines instructions ont à cet égard un comportement propre qui sera décrit pour chacune d'elles. Ce sont BRX et TES.

• Le mode : indique dans quel mode de travail doit être l'unité centrale pour autoriser l'exécution de certaines instructions. Une instruction dite "privilégiée" devra être exécutée en mode maître. Par ailleurs elle donnera lieu à un cas de déroutement supplémentaire : violation de mode.

• L'option. Certaines instructions ne figurent pas obligatoirement dans toutes les unités centrales. Elles seront dites optionnelles. Elles donneront lieu à un cas de déroutement supplémentaire : Instruction inexistante.

Cependant, les instructions SHC, DIV, FAD, FSU, FMU et FDV peuvent être traitées par un module moniteur les simulant (module TRAP).

VII-2. NOTATIONS SYMBOLIQUES UTILISEES

Pour la description de la fonction des instructions

A — Registre accumulateur

$\overline{(A)}$ — Complément logique de A

A_{0-7} — Octet de poids forts de l'accumulateur

A_{8-15} — Octet de poids faibles de l'accumulateur

C — Indicateur Carry (Report)

D — Déplacement (octet de poids faibles, de l'instruction, étendu à un mot par adjonction d'un octet de poids nul. Soit :

xxxxxxxxyyyyyyyy ← Instruction

00000000yyyyyyyy ← Déplacement

E — Extension de l'accumulateur

E, A — Registre étendu composé de l'accumulateur et de son extension, les poids forts étant dans E.

G — Base générale

G'	En mode esclave G' = G En mode maître G' = O
L	Base locale
MA	Indicateur de masquage des interruptions
MS	Indicateur de mode (master-slave ≡ maître-esclave)
N	Opérande calculé. C'est le mot contenu à l'adresse calculée Y
N_{11-15}	Contenu des bits 11 à 15 de l'opérande calculé
O	Indicateur Overflow (Débordement)
P	Base programme
PR	Indicateur de protection
Rn	Registre de numéro n
X	Registre d'index
Y	Adresse calculée (Déplacement traité par le mode d'adressage)
Y_2	Adresse mot correspondant à y_2 Si Y est paire $Y_2 = Y$ Si Y est impaire $Y_2 = Y - 1$
α	Rang courant dans une chaîne d'octets ou de mots traitée par une instruction, le premier octet de la chaîne ayant le rang zéro.
bp	Bit de protection
r	Contenu du registre R
rn	Contenu du registre Rn
y	Octet d'adresse Y
y_2	Mot d'adresse Y_2
( )	Contenu de. Exemple $y_2 = (Y_2)$ $N = (Y_2)$
⟵	Remplace
⟷	Echangés
#	Modifié mais non significatif
⊕	ou exclusif
∧	"et" logique
∨	"ou" logique

■ Dans les exemples

Pour familiariser l'utilisateur avec l'écriture en langage d'assemblage, quelques exemples sont fournis avec chaque instruction, donnant un éventail varié des possibilités qui lui

permettra d'aborder la programmation sans connaître complètement tous les cas permis par l'assembleur.

Pour simplifier la représentation de ces exemples, les normes suivantes ont été utilisées:

Les identificateurs sont composés de quatre lettres et d'un numéro d'ordre. Le sens de ces lettres est le suivant :

1ère lettre	E	Etiquette
	S	Symbole (défini par une directive EQU)
2ème lettre	P	Prédéfini
	A	Anticipé
3ème lettre	D	Défini dans un segment de données
	P	Défini dans un segment de programme
4ème lettre	C	Commun
	L	Local

Exemple :

EPDC27 Etiquette prédéfinie de données communes

Tous les exemples sont ainsi écrits relativement à un CDS et un LDS ayant la forme suivante :

```
EXEMPL          CDS

EPDC1           DATA        7
EPDC2           DATA        EPDC1
EPDC3           DATA        EPDC4
EPDC4           RES         20
EPDC5           DATA,1      &FF,00
EPDC6           DATA        EPDC5
EPDC7           DATA        EPDC8
EPDC8           RES,1       10
EPDC9           DATA        "AB"
                DATA        "CD"
EPDC10          DATA        EPDC9
EPDC11          DATA        6
EPDC12          DATA        &82
EPDC13          DATA        EPPL2
EPDC14          DATA        EPPL3
EPDC15          DATA        EPPL1
                DATA        EPPL2
                DATA        EPPL3
                DATA        EPPL4
EPDC16          TEXT        "ABCDEFGHIJ"
EPDC17          DATA        EPDC16
EPDC18          DATA,1      "A"
                DATA,1      "E"
                DATA,1      "X"
EPDC19          DATA        EPDC18
```

```
EPDC20      DATA    EPDC21
            DATA    EPDC21 + 2
            DATA    EPDC21 + 4
EPDC21      DATA    3
            DATA    4
            DATA    5
EPDC22      DATA    EPDC23
            DATA    EPDC23 + 2
            DATA    EPDC23 + 4
            DATA    EPDC23 + 6
            DATA    EPDC23 + 8
EPDC23      RES     10
EPDC24      DATA    EPDC25
            DATA    EPDC25 + 1
            DATA    EPDC25 + 2
            DATA    EPDC25 + 3
EPDC25      RES,1   5
EPDC26      DATA    &30A2
            DATA    &400B
            DATA    &40FA
            DATA    &BF01
EPDC27      DATA    EPD26
            DATA    EPDC26 + 4
EPDC28      RES     5
EPDC29      DATA    EPDC28
SPDC1       EQU     EPDL1
SPDC2       EQU     3
SPDC3       EQU     SPL2
SPDC4       EQU     EPDL14
            FIN


LDS1        LDS

EPDL1       DATA    7
EPDL2       DATA    EPPL1
EPDL3       DATA    EPDL4
EPDL4       RES     20
EPDL5       DATA,1  &FF,00
EPDL6       DATA    EPDL5
EPDL7       DATA    EPDL6
EPDL8       RES,1   10
EPDL9       DATA    "AB"
            DATA    "CD"
EPDL10      DATA    EPDL9
EPDL11      DATA    6
EPDL12      DATA    &82
EPDL13      DATA    EPPL2
EPDL14      DATA    EPPL3
```

```
EPDL15      DATA        EPPL1
            DATA        EPPL2
            DATA        EPPL3
            DATA        EPPL4
EPDL16      TEXT        "ABCDEFGHIJ"
EPDL17      DATA        EPDL16
EPDL18      DATA,1      "A"
            DATA,1      "E"
            DATA,1      "X"
EPDL19      DATA        EPDL18
EPDL20      DATA        EPDL21
            DATA        EPDL21 + 2
            DATA        EPDL21 + 4
            DATA        3
            DATA        4
            DATA        5
EPDL22      DATA        EPDL23
            DATA        EPDL23 + 2
            DATA        EPDL23 + 4
            DATA        EPDL23 + 6
            DATA        EPDL23 + 8
EPDL23      RES         10
EPDL24      DATA        EPDL25
            DATA        EPDL25 + 1
            DATA        EPDL25 + 2
            DATA        EPDL25 + 3
EPDL25      RES,1       5
EPDL26      DATA        &30A2
            DATA        &400B
            DATA        &40FA
            DATA        &BF01
EPPL27      DATA        EPPL26
            DATA        EPDL26 + 4
EPDL28      RES         5
EPDL29      DATA        EPDL28
SPDL1       EQU         EPDL1
SPDL2       EQU         3
SPDL3       EQU         SPDL2
SPDL4       EQU         EPDL14
            FIN
LPS1        LPS         LDS1
             .
             .
             .      Instructions
             .
             .
             .
            FIN         EPPL1
            END         LPS1
```

VII-3. CHARGEMENTS ET RANGEMENTS

VII-3.1. Introduction

Les chargements et rangements sont représentés dans le tableau suivant et seront détaillés ensuite instruction par instruction.

	Octet		Mot		Double mot	
	Chargement	Rangement	Chargement	Rangement	Chargement	Rangement
Données	LBL LBR LBX	SBL SBR	LDA LDE LDX LDR	STA STE STX STR	DLD	DST
Adresse			LEA	SPA		
Sélectif				STS		

III-3.2. Description

L'ordre suivi dans la description est le suivant :

Octet

- LBL (Load Byte Left) Chargement de l'octet gauche de A
- SBL (Store Byte Left) Rangement de l'octet gauche de A
- LBR (Load Byte Right) Chargement de l'octet droit de A
- SBR (Store Byte Right) Rangement de l'octet droit de A
- LBX (Load Byte X) Chargement de l'octet droit de X

Mot

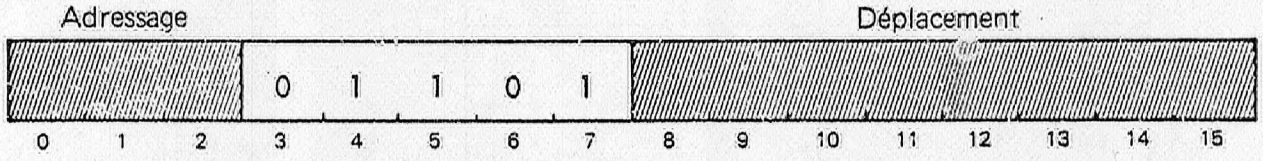
- LDA (Load A) Chargement du registre A
- STA (Store A) Rangement du registre A
- LDE (Load E) Chargement du registre E
- STE (Store E) Rangement du registre E
- LDX (Load X) Chargement du registre X
- STX (Store X) Rangement du registre X
- LDR (Load Register) Chargement d'un registre
- STR (Store Register) Rangement d'un registre
- LEA (Load Effective Address) Chargement de l'adresse effective

Mot	SPA	(Store Program Address) Rangement de l'adresse programme
	STS	(Store Selective) Rangement sélectif de A
Double mot	DLD	(Double Load) Chargement du registre étendu E, A
	DST	(Double Store) Rangement du registre étendu E, A

NOM : Chargement de l'octet gauche de A (Load Byte Left) **LBL**

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	0D	2,6
P	2D	3
DG	4D	2,6
IL	6D	3,6
IGX	8D	3,6
ILX	AD	3,5

Fonction : $y \longrightarrow (A_{0-7})$

(A_{8-15}) inchangé

L'octet de poids forts du registre A est chargé avec la valeur lue en mémoire vive à l'adresse Y.

L'octet de poids faible du registre A reste inchangé.

Eléments modifiés :

- Registres : A_{0-7}
- Positions mémoire :
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0	0	(A) > 0
0	1	(A) < 0
1	0	(A) = 0
1	1	

Déroutements : standard

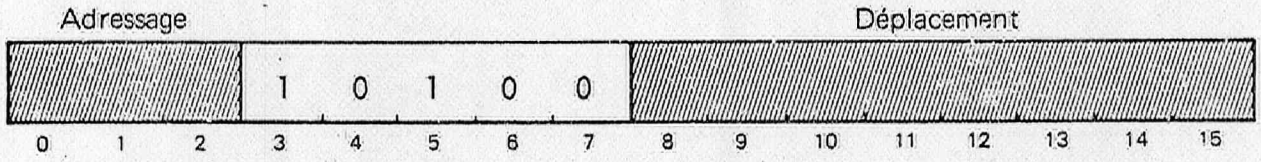
Exemples :

```
LBL     EPDL1 + 1
LBL     = 0
LBL     #EPDC1
LBL     @ EPDL6
LBL     @#EPDC7,x
LBL     @ EPDL7,x
```

NOM : Rangement de l'octet gauche de A (Store Byte Left) **SBL**

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	14	2,5
DG	54	2,5
IL	74	3,5
IGX	94	3,5
ILX	B4	3,5

Fonction : $(A_{0-7}) \longrightarrow y$
(A) inchangé

Le contenu de l'octet gauche (octet de poids fort) du registre A est rangé à l'adresse Y de la mémoire vive.

Eléments modifiés :

- Positions mémoire : y

Déroutements : standard

Exemples :

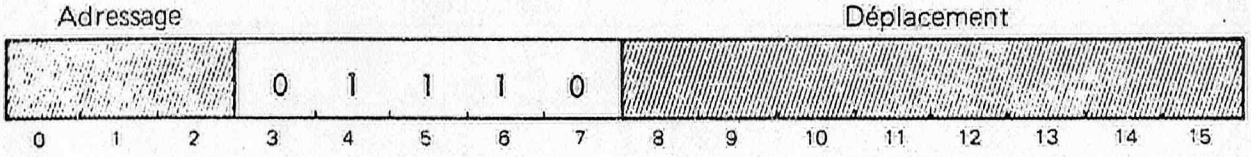
```
SBL     EPDL8
SBL     #EPDC8 + 3
SBL     @ EPDL24
SBL     @#EPDC24,x
SBL     @ EPDL24,x
```

LBR

NOM : Chargement du registre droit de A (Load Byte Right

Classe : 0 Privilégiée : non Optionnelle : no

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	0E	2,6
P	2E	3
DG	4E	2,6
IL	6E	3,6
IGX	8E	3,6
ILX	AE	3,6

Fonction : $y \longrightarrow (A_{8-15})$

$0 \longrightarrow (A_{0-7})$

L'octet de poids faible du registre A est chargé avec la valeur lue en mémoire vive à l'adresse Y.

L'octet de poids fort du registre A est remis à zéro.

Eléments modifiés :

- Registre A
- Positions mémoire :
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0	0	(A) > 0
0	1	
1	0	(A) = 0
1	1	

Déroutements : standard

Exemples :

```
LBR     SPDL4
LBR     =SPDL2
LBR     #EPDC1
LBR     @ EPDL6
LBR     @ #EPDC7,x
LBR     @ EPDL7,x
```

SBR

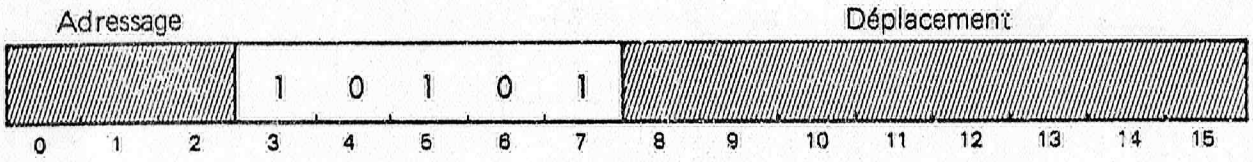
NOM : Rangement de l'octet droit de A (Store Byte Right)

Classe : 0'

Privilégiée : non

Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	15	2,5
DG	55	2,5
IL	75	3,5
IGX	95	3,5
ILX	B5	3,5

Fonction : $(A_{8-15}) \longrightarrow y$
(A) inchangé

Le contenu de l'octet droit (octet de poids faible) du registre A est rangé à l'adresse Y de la mémoire vive.

Eléments modifiés :

- Positions mémoire : y

Déroutements : standard

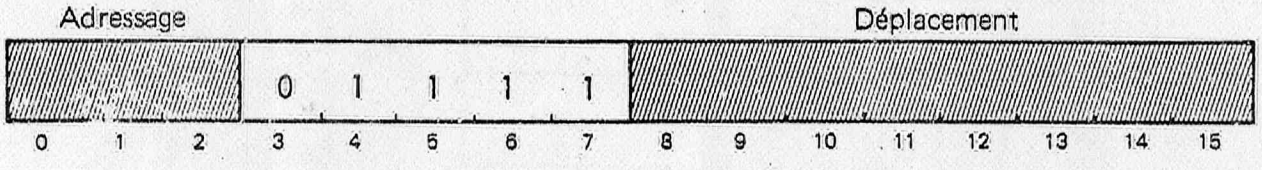
Exemples :

```
SBR     EPDL8 + 3
SBR     #EPDC8
SBR     @EPDL7
SBR     @#EPDC24,x
SBR     @#EPDL24,x
```

NOM : Chargement de l'octet droit de X (Load Byte X) **LBX**

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	0F	2,6
P	2F	3
DG	4F	2,6
IL	6F	3,6
IGX	8F	3,6
ILX	AF	3,6

Fonction : $y \longrightarrow (X_{8-15})$

$0 \longrightarrow (X_{0-7})$

L'octet de poids faible du registre X est chargé avec la valeur lue en mémoire vive à l'adresse Y.

L'octet de poids fort du registre X est remis à zéro.

Eléments modifiés :

- Registre : X
- Positions mémoire :
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0	0	(X) > 0
0	1	
1	0	(X) = 0
1	1	

Déroutements : standard

Exemples :

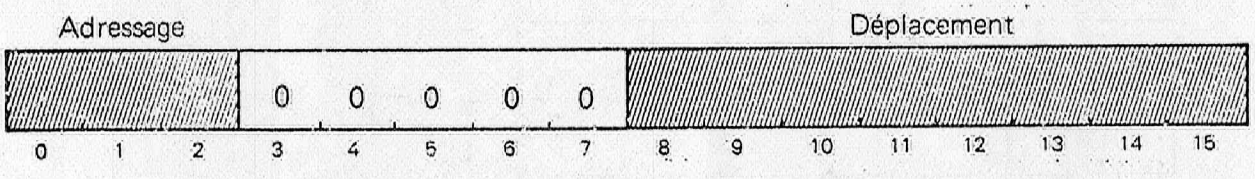
```
LBX     EPDL5
LBX     =1
LBX     #EPDC5
LBX     @ EPDL6
LBX     @#EPDC7,x
LBX     @ EPDL7,x
```

NOM : Chargement du registre (LoaD A)

LDA

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	00	2,3
P	20	2,5
DG	40	2,2
IL	60	3,4
IGX	80	3,4
ILX	A0	3,4

Fonction : $y_2 \longrightarrow (A)$

Le registre A est chargé avec la valeur lue en mémoire vive à l'adresse Y_2 .

Eléments modifiés :

- Registre : A
- Positions mémoire :
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0	0	(A) > 0
0	1	(A) < 0
1	0	(A) = 0
1	1	

Déroutements : standard

Exemples :

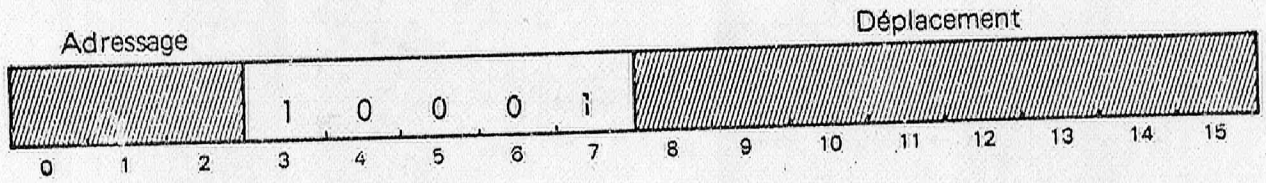
```
LDA     EPDL1
LDA     =/
LDA     #EPDC1
LDA     @ EPDL2
LDA     @ #EPDC3,x
LDA     @ EPDL3,x
```

STA

NOM : Rangement du registre A (STore A)

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	11	2,2
DG	51	2,2
IL	71	3,2
IGX	91	3,2
ILX	B1	3,2

Fonction : (A) $\longrightarrow$ Y_2
(A) inchangé

Le contenu du registre A est rangé à l'adresse Y_2 de la mémoire vive.

Eléments modifiés :

- Positions mémoire : y_2

Déroutements : standard

Exemples :

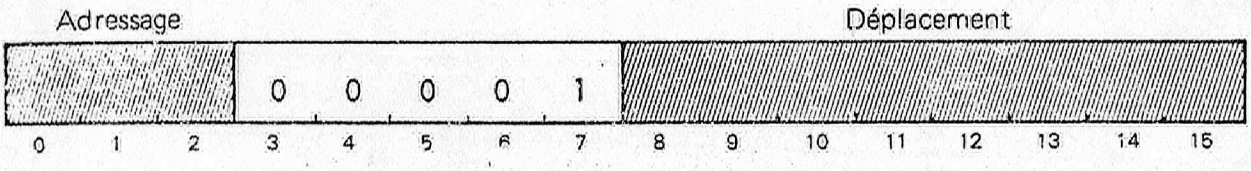
```
STA     EPDL23 + 4
STA     #EPDC23
STA     @EPDL3
STA     @#EPDC22,x
STA     @EPDL22,x
```

LDE

NOM : Chargement du registre E (LoaD E)

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	01	2,3
P	21	2,5
DG	41	2,3
IL	61	3,4
IGX	81	3,4
ILX	A1	3,4

Fonction : $y_2 \longrightarrow (E)$

Le registre E est chargé avec la valeur lue en mémoire vive à l'adresse Y_2.

Eléments modifiés :

- Registres : E
- Positions mémoire :
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0	0	(E) > 0
0	1	(E) < 0
1	0	(E) = 0
1	1	

Déroutements : standard

Exemples :

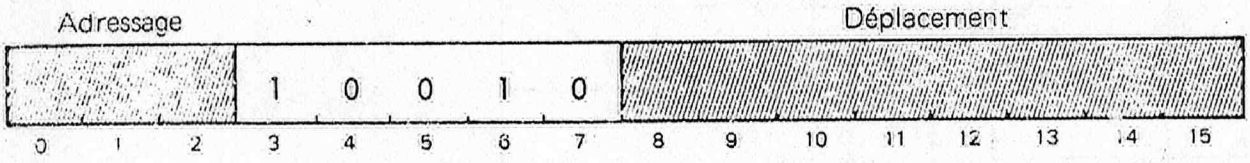
```
LDE     SPDL1
LDE     =&FF
LDE     #SPDC1
LDE     @ EPDL3
LDE     @ #EPDC3,x
LDE     @ EPDL3,x
```

STE

NOM : Rangement du registre E (STore E)

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	12	2,2
DG	52	2,2
IL	72	3,2
IGX	92	3,2
ILX	B2	3,2

Fonction : (E) $\longrightarrow Y_2$
(E) inchangé

Le contenu du registre E est rangé à l'adresse Y_2 de la mémoire vive.

Eléments modifiés :

- Positions mémoire : Y_2

Déroutements : standard

Exemples :

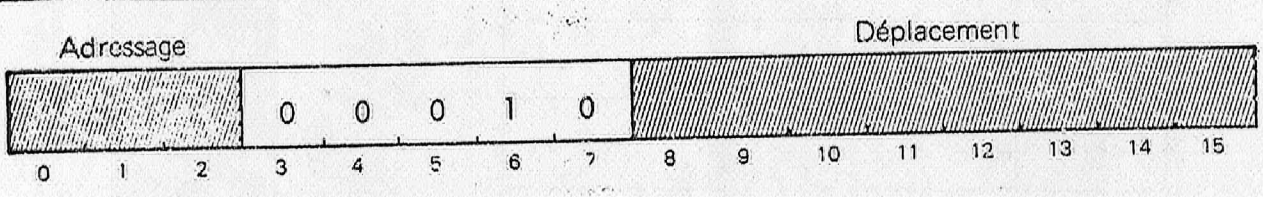
```
STE     EPDL4
STE     #EPDC4 + 2
STE     @ EPDL3
STE     @#EPDC22,x
STE     @ EPDL22,x
```

LDX

NOM : Chargement du registre X (LoaD X)

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	02	2,3
P	22	2,5
DG	42	2,3
IL	62	3,4
IGX	82	3,4
ILX	A2	3,4

Fonction : $Y_2 \longrightarrow (X)$

Le registre X est chargé avec la valeur lue en mémoire vive à l'adresse Y_2 .

Eléments modifiés :

- Registres : X
- Positions mémoire :
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0	0	(X) > 0
0	1	(X) < 0
1	0	(X) = 0
1	1	

Déroutements : standard

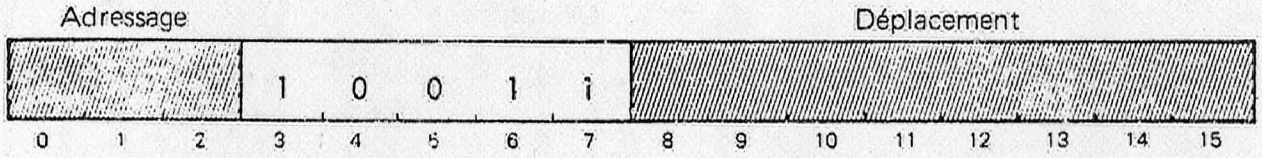
Exemples :

```
LDX     EPDL2
LDX     =4
LDX     #EPDC1
LDX     @EPDL3
LDX     @#EPDC3,x
LDX     @EPDL3,x
```

NOM : Rangement du registre X (STore X) **STX**

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	13	2,2
DG	53	2,2
IL	73	3,2
IGX	93	3,2
ILX	B3	3,2

Fonction : (X) $\longrightarrow$ Y_2
(X) inchangé

Le contenu du registre X est rangé à l'adresse Y_2 de la mémoire vive.

Eléments modifiés :

- Position mémoire : Y_2

Déroutements : standard

Exemples :

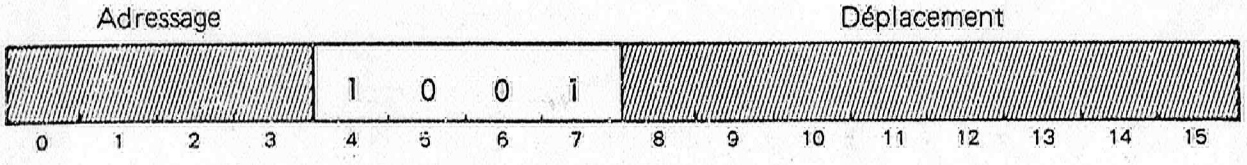
```
STX     EPDL23 + 6
STX     #EPDC23 + 2
STX     @EPDL22
STX     @#EPDC22,x
STX     @EPDL22,x
```

LDR

NOM : Chargement d'un registre (LoaD Register)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	39	3,7
PX	E9	3,7
P	F9	3,7

Fonction : $(R_n) \longrightarrow A$
(R_n) inchangé
avec n = (Y)
Le registre A est chargé avec la valeur contenue dans le registre adressé numéro n

Eléments modifiés :

- Registres : A
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0	0	(A) > 0
0	1	(A) < 0
1	0	(A) = 0
1	1	

Déroutements : standard

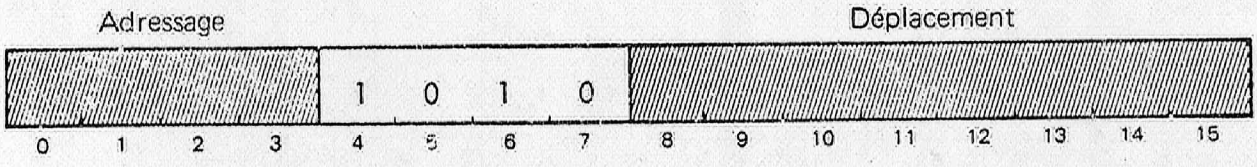
Exemples :

```
LDR     EPDL1
LDR     =0,x
LDR     =22
```

NOM : Rangement d'un registre (STore Register) **STR**

Classe : 1 Privilégiée : oui Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code hexadécimal	Temps d'exécution en µs
DL	3A	4
PX	EA	4
P	FA	4

Fonction : (A) ⟶ (R_n)
(A) inchangé
avec n = (Y)
Le contenu du registre A est rangé dans le registre adressé numéro n.

Eléments modifiés :

- Registres : R_n

Déroutements :

- Standard et violation de mode

Divers :

Cette instruction ne peut être utilisée qu'en mode maître car elle modifie l'état d'un registre quelconque (en particulier les registres d'Entrée/Sortie). Elle est normalement réservée aux moniteurs et systèmes d'exploitation.

Exemples :

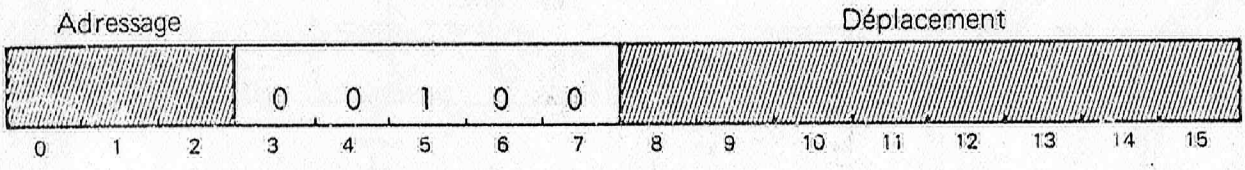
```
STR     EPDL1
STR     =2,x
STR     =4
```

LEA

NOM : Chargement adresse effective (Load Effective Address)

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
		Mode maître	Mode esclave
DL	04	2,7	3,3
P	24	3,1	3,7
DG	44	2,7	3,3
IL	64	3,7	4,3
IGX	84	3,7	4,3
ILX	A4	3,7	4,3

Fonction : Y - (G) ⟶ A

L'adresse calculée est décrémentée du contenu du registre G. Le résultat, représentant l'adresse effective, est rangé dans le registre A.

Eléments modifiés :

- Registres : A

Déroutements : standard

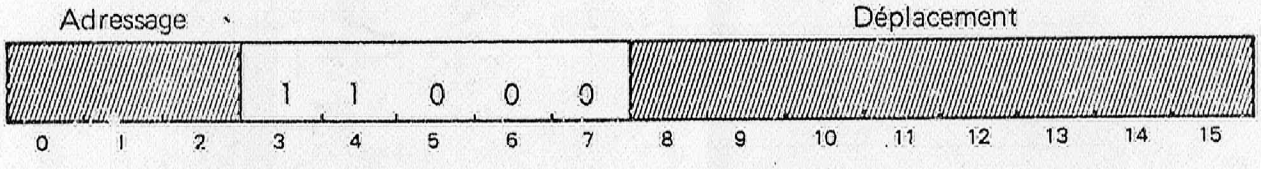
Exemples :

```
LEA     EPDC7
LEA     =&2F
LEA     #EPDL11
LEA     @EPDC2
LEA     @#EPDC22,x
LEA     @EPDL22
```

NOM : Rangement adresse programme (Store Program Address) **SPA**

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
		Mode maître	Mode esclave
DL	18	3,1	3,4
DG	58	3,1	3,4
IL	78	4,2	4,5
IGX	98	4,2	4,5
ILX	B8	4,2	4,5

Fonction : $(P) + 4 - G' \longrightarrow y_2$

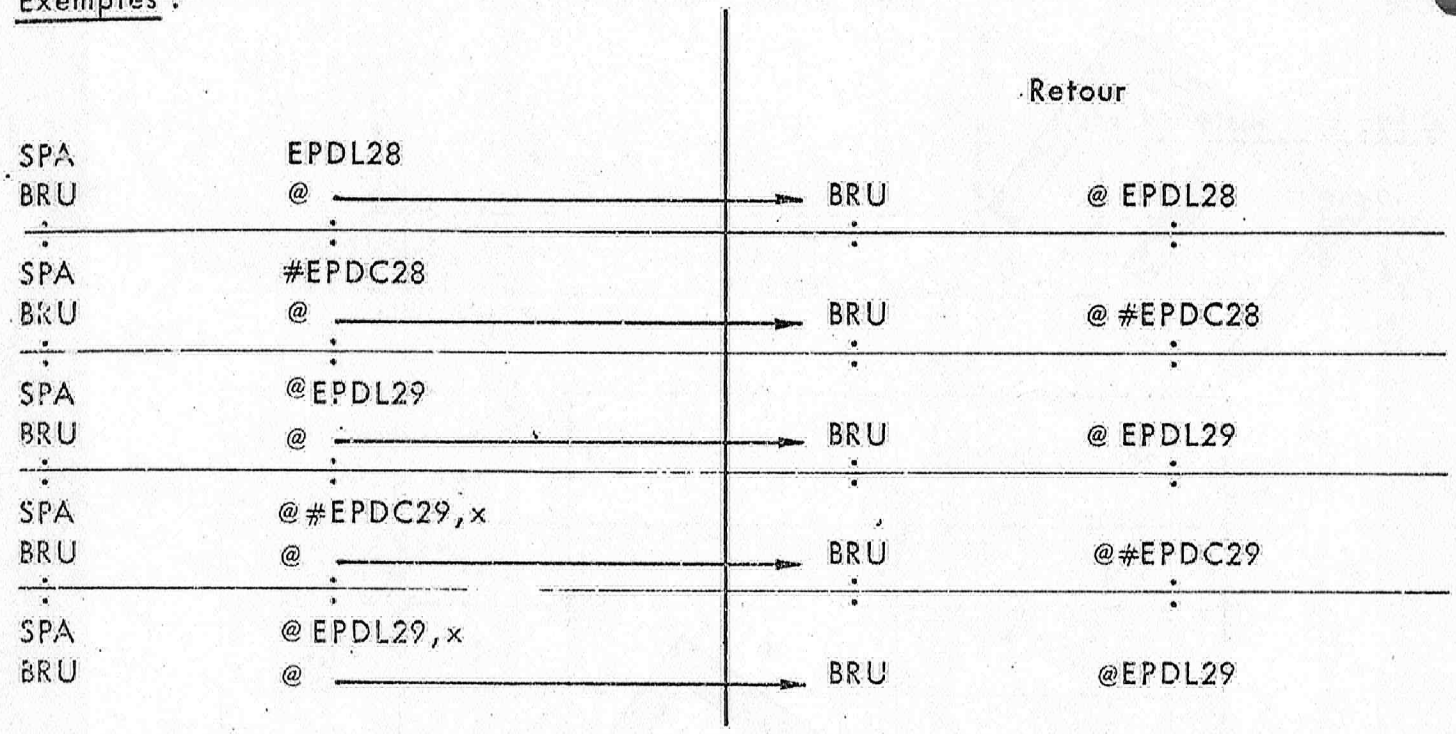
Cette instruction est suivie normalement par un branchement indirect à un sous-programme. En conséquence l'adresse courante (contenu du registre P) augmentée de quatre est rangée dans le registre A. Cette adresse augmentée de quatre constitue normalement l'adresse de retour du sous-programme.

Eléments modifiés :

- Registres :
- Positions mémoire : y_2
- Indicateurs :

Déroutements : standard

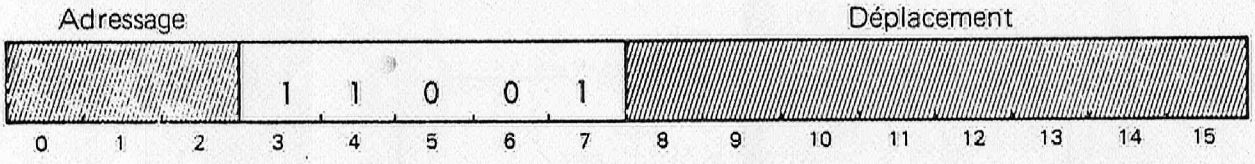
Exemples :



NOM : Rangement sélectif de A (STore Selective) **STS**

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :

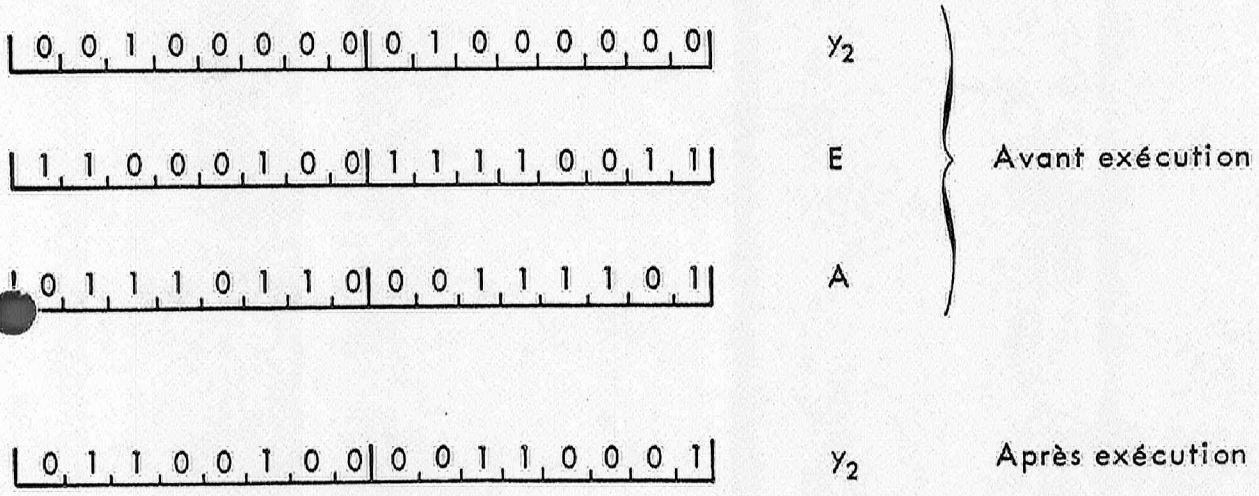


Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	19	3,4
DG	59	3,4
IL	79	4,4
IGX	99	4,4
ILX	B9	4,4

Fonction : $y_2 \wedge (\overline{E}) \vee (A) \wedge (E) \longrightarrow y_2$

(A) et (E) inchangés

Les bits du mot situé à l'adresse Y_2 de la mémoire vive, correspondant aux bits à 1 du registre E sont chargés avec les bits correspondants du registre A. Les autres bits du mot situé à l'adresse Y_2 n'étant pas modifiés. Soit par exemple :



Eléments modifiés :

- Registres :

- Positions mémoire : y_2

- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0	0	Résultat > 0
0	1	Résultat < 0
1	0	Résultat = 0
1	1	

Déroutements : standard

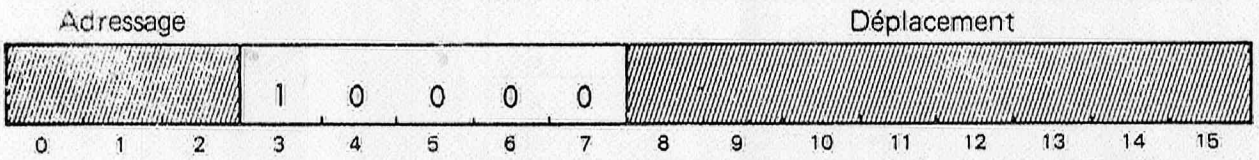
Exemples :

```
STS     EPDL4
STS     #EPDC23
STS     @EPDL3
STS     @#EPDC22,x
STS     @EPDL22,x
```

NOM : Chargement des registres A et E (Double LoaD) **DLD**

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'éxecution en µs
DL	10	3,6
DG	50	3,6
IL	70	4,8
IGX	90	4,8
ILX	B0	4,8

Fonction : $(Y_2) \longrightarrow (E)$

$(Y_2 + 2) \longrightarrow (A)$

Le registre E est chargé avec la valeur lue en mémoire vive à l'adresse Y_2, et le registre A est chargé avec la valeur lue en mémoire vive à l'adresse $Y_2 + 2$.

Eléments modifiés :

- Registres : E-A
- Positions mémoire :
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	(E) > 0
0	1	(E) < 0
1	0	(E) = 0
1	1	

Déroutements : standard

Exemples :

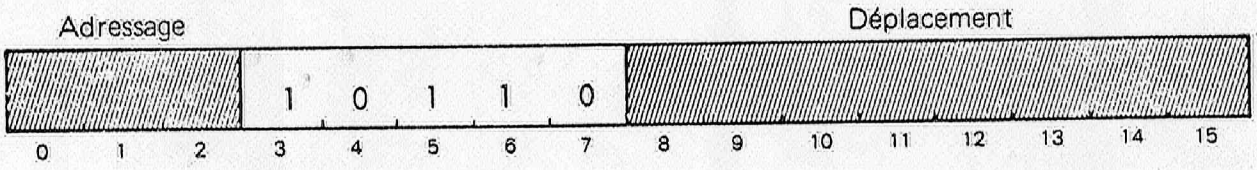
```
DLD     EPDL9
DLD     #EPDC9
DLD     @EPDL10
DLD     @#EPDC3,x
DLD     @EPDL3,x
```

NOM : Rangement des registres E et A (Double STore)

DST

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	16	3,6
DG	56	3,6
IL	76	4,7
IGX	96	4,7
ILX	B6	4,7

Fonction : (E) → (Y_2)

(A) → (Y_2 + 2)

(A) et (E) inchangés

Le contenu du registre E est rangé à l'adresse Y_2 de la mémoire vive, et le contenu du registre A est rangé à l'adresse Y_2 + 2 de la mémoire vive.

Eléments modifiés :

- Registres :
- Positions mémoire : (Y_2) et (Y_2 + 2)
- Indicateurs

Déroutements : standard

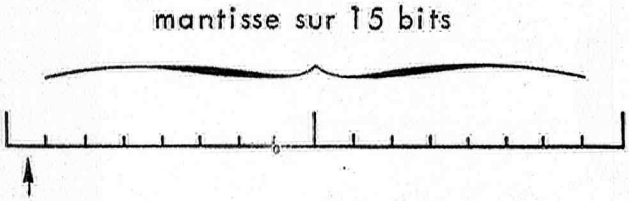
Exemples :

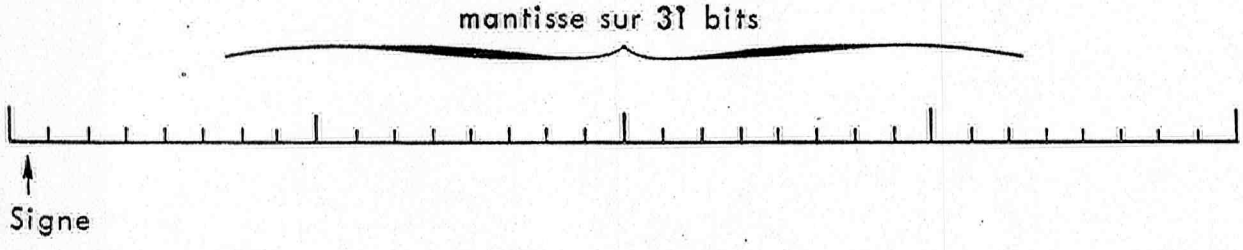
```
DST     EPDL4
DST     #EPDC4
DST     @EPDL3
DST     @#EPDC22,x
DST     @EPDL22,x
```

VII-4. ARITHMETIQUE VIRGULE FIXE

VII-4.1. Introduction

Les opérandes sont considérés comme des nombres entiers signés sur un ou deux mots.





Les nombres négatifs sont représentés par le complément à 2 de leur valeur absolue.

VII-4.2. Description

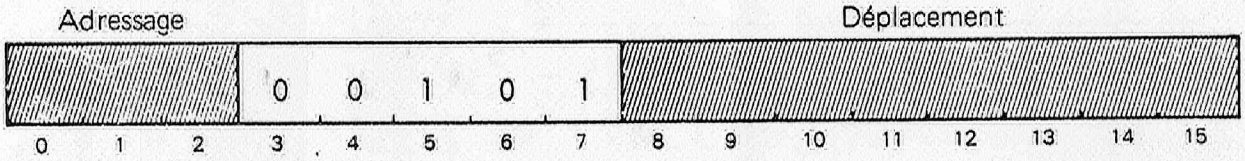
Les instructions traitant l'arithmétique virgule fixe permettent l'exécution des quatre opérations et sont décrites dans la suite dans l'ordre suivant :

ADD (Addition)	Addition
ADM (Addition Memory)	Addition mot mémoire
SUB (Substraction)	Soustraction
MUL (Multiplication)	Multiplication
DIV (Division)	Division

NOM : Addition (ADDition) **ADD**

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	05	2,3
P	25	2,5
DG	45	2,3
IL	65	3,4
IGX	85	3,4
ILX	A5	3,4

Fonction : $(A) + y_2 \longrightarrow (A)$

La valeur lue en mémoire vive à l'adresse Y_2 est additionnée au contenu du registre A. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Positions mémoire :
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
1	-	Report
-	1	Débordement

Déroutements : standard

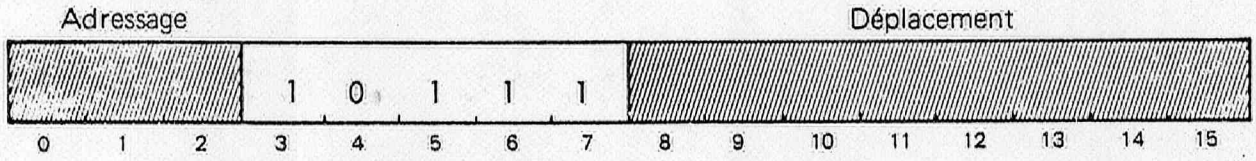
Exemples :

```
ADD        EPDL12
ADD        =122
ADD        #EPDC12
ADD        @EPDL2
ADD        @#EPDC20,x
ADD        @EPDL20,x
```

NOM : Addition mot mémoire (ADition Memory) **ADM**

Classe : 0' Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	17	2,6
DG	57	2,6
IL	77	3,7
IGX	97	3,7
ILX	B7	3,7

Fonction : $y_2 + (A) \longrightarrow Y_2$ et A

Le contenu du registre A est additionné à la valeur lue en mémoire vive à l'adresse Y_2. Le résultat est rangé dans le mot situé à l'adresse Y_2 et dans le registre A.

Eléments modifiés :

- Registres : A
- Positions mémoire : y_2
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
-	1	Débordement
1	-	Report

Déroutements : standard

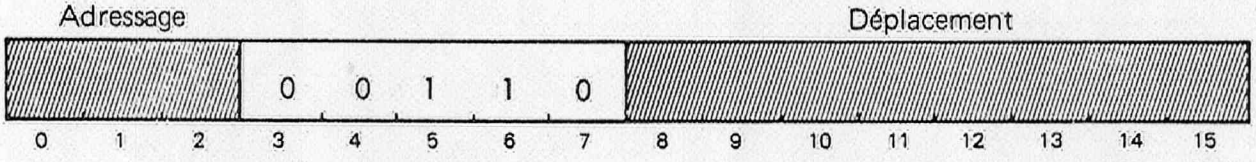
Exemples :

```
ADM     EPDL4
ADM     #EPDC4
ADM     @EPDL3
ADM     @#EPDC20,x
ADM     @EPDL20,x
```

NOM : Soustraction (SUBstraction) **SUB**

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	06	2,3
P	26	2,5
DG	46	2,3
IL	66	3,4
IGX	86	3,4
ILX	A6	3,4

Fonction : (A) - y_2 ⟶ (A)

La valeur lue en mémoire vive à l'adresse Y_2 est soustraite du contenu du registre A. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Positions mémoire :
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
1	-	Report
-	1	Débordement

Déroutements : standard

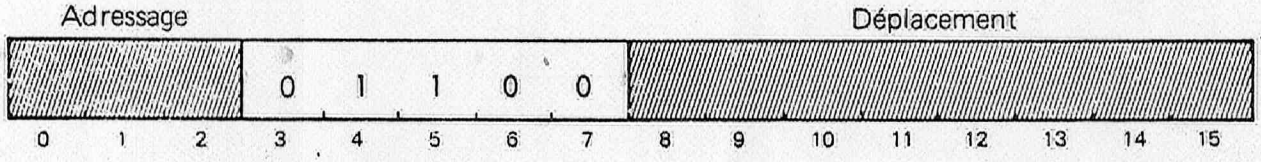
Exemples :

```
SUB     EPDL11
SUB     =2
SUB     #EPDC11
SUB     @EPDL20
SUB     @#EPDC20,x
SUB     @EPDL20,x
```

NOM : Multiplication algébrique (MULtiplication) **MUL**

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
DL	0C	8,1	38,4+0,9 n
P	2C	8,5	38,8+0,9 n
DG	4C	8,1	38,8+0,9 n
IL	6C	9,1	39,4+0,9 n
IGX	8C	9,1	39,4+0,9 n
ILX	AC	9,1	39,4+0,9 n
		Câblée	Microprogrammée

n = nombre de positions à 1 du multiplicateur.

Fonction : $(A) \times y_2 \longrightarrow (E,A)$

La valeur lue en mémoire vive à l'adresse Y_2 est multipliée algébriquement par le contenu du registre A. Le résultat est rangé dans les registres E et A.

Le registre E contient les poids forts du résultat et le registre A les poids faibles.

Eléments modifiés :

- Registres : E - A
- Positions mémoire :
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	E > 0
0	1	E < 0
1	0	E = 0

Déroutements : standard

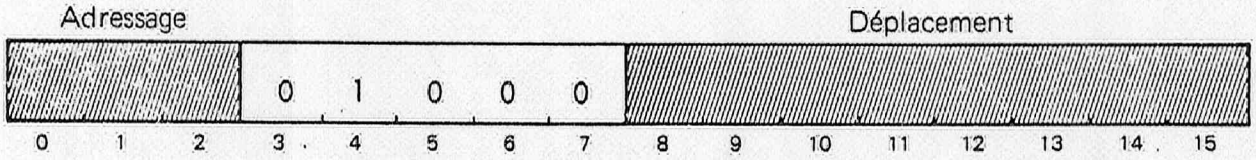
Exemples :

```
MUL     EPDL11
MUL     =&2E
MUL     #EPDC1
MUL     @EPDL20
MUL     @#EPDC20,x
MUL     @EPDL20,x
```

NOM : Division algébrique (DIVision) **DIV**

Classe : 0 Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
DL	08	9,1	42 +0,3 n
P	28	9,5	42,4 +0,3 n
DG	48	9,1	42 +0,3 n
IL	68	10,1	43 +0,3 n
IGX	88	10,1	43 +0,3 n
ILX	A8	10,1	43 +0,3 n

Câblée Microprogrammée

n = nombre de positions à 1 du quotient.

Fonction : (E,A) : y_2 ⟶ (A)

reste ⟶ E

le reste est du signe du dividende (sauf si le reste est nul, auquel cas le signe est

Le contenu de l'accumulateur (Registre A contenant les poids faibles) et de son extension (Registre E contenant les poids forts) est divisé par la valeur lue en mémoire vive à l'adresse Y_2. Le quotient est rangé dans le registre A et le reste dans le registre E.

Le signe du reste est égal au signe initial du registre E, sauf si le reste est égal à zéro auquel cas le signe est +.

Eléments modifiés :

- Registres : E-A (si division par zéro ou débordement ils sont inchangés).

- Positions mémoire :

- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
#	1	Débordement ou division par zéro

Déroutements : Instruction inexistante et standard

Exemples :

```
DIV     EPDL1
DIV     =5
DIV     #EPDC11
DIV     @EPDL2
DIV     @#EPDC20,x
DIV     @EPDL20,x
```

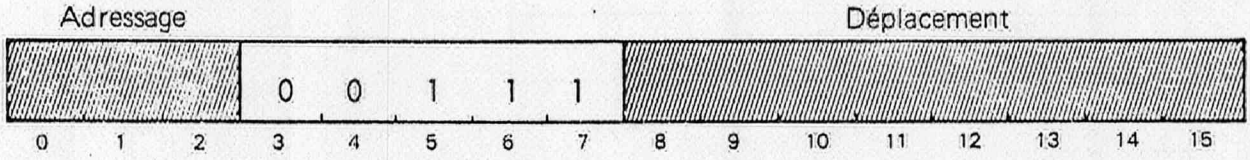
VII-5. OPERATIONS LOGIQUES

IOR	(Inclusive OR)	Réunion
EOR	(Exclusive OR)	Disjonction
AND	(AND)	Intersection
CMP	(Compare)	Comparaison

NOM : Réunion (Inclusive OR) **IOR**

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	07	2,3
P	27	2,5
DG	47	2,3
IL	67	3,4
IGX	87	3,4
ILX	A7	3,4

Fonction : $(A) \vee y_2 \longrightarrow (A)$

Réunion (OU inclusif) entre la valeur lue en mémoire vive à l'adresse Y_2 et le contenu du registre A. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Positions mémoire :
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	(A) > 0
0	1	(A) < 0
1	0	(A) = 0

Déroutements : standard

Exemples :

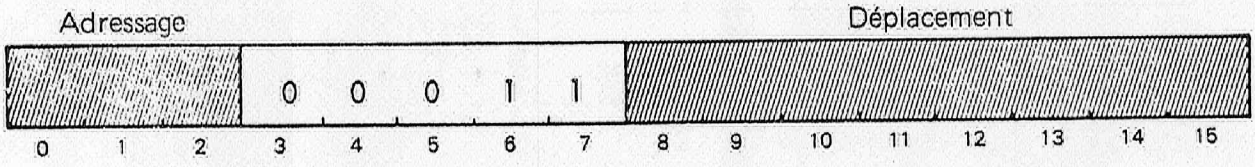
```
IOR     EPDL21
IOR     =&01
IOR     #EPDL9
IOR     @EPDL20
IOR     @#EPDC20,x
IOR     @EPDL20,x
```

NOM : Disjonction (Exclusive OR)

EOR

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	03	2,3
P	23	2,5
DG	43	2,3
IL	63	3,4
IGX	83	3,4
ILX	A3	3,4

Fonction : (A) $\oplus$ $y_2 \longrightarrow$ (A)

Disjonction (OU exclusif) entre la valeur lue en mémoire vive à l'adresse Y_2 et le contenu du registre A. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Positions mémoire :
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	(A) > 0
0	1	(A) < 0
1	0	(A) = 0

Déroutements : standard

Exemples :

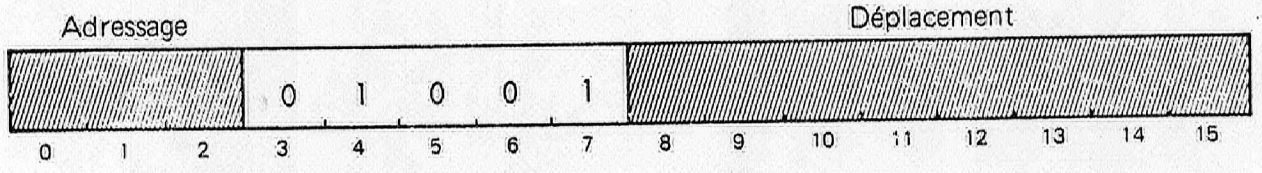
```
EOR     EPDL1
EOR     =&80
EOR     #EPDC1
EOR     @EPDL2
EOR     @#EPDC20,x
EOR     @EPDL20
```

AND

NOM : Intersection (AND)

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	09	2,3
P	29	2,5
DG	49	2,3
IL	69	3,4
IGX	89	3,4
ILX	A9	3,4

Fonction : $(A) \wedge y_2 \longrightarrow (A)$

Intersection (ET) entre la valeur lue en mémoire vive à l'adresse Y_2 et le contenu du registre A. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Positions mémoire :
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	(A) > 0
0	1	(A) < 0
1	0	(A) = 0

Déroutements : standard

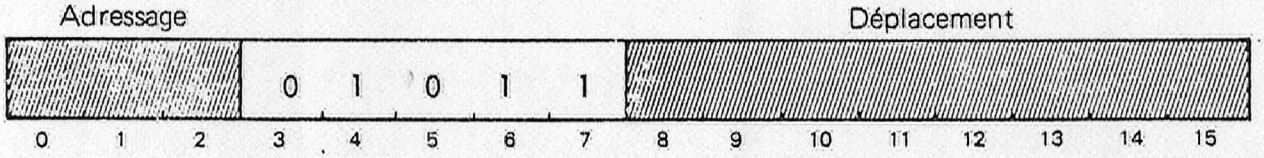
Exemples :

```
AND     EPDL1
AND     =&0F
AND     #EPDC1
AND     @EPDL20
AND     @#EPDC20,x
AND     @EPDL20,x
```

NOM : Comparaison (CoMPare) **CMP**

Classe : 0 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code hexadécimal	Temps d'exécution en µs		
		>	<	=
DL	0B	4,1	4,5	3,2
P	2B	4,5	5	3,6
DG	4B	4,1	4,5	3,2
IL	6B	5,1	5,6	4,2
IGX	8B	5,1	5,6	4,2
ILX	AB	5,1	5,6	4,2

Fonction : (A) comparé algébriquement à y_2

Résultat dans les indicateurs

Le contenu du registre A est comparé algébriquement à la valeur lue en mémoire vive à l'adresse Y_2. Le résultat est rangé dans les indicateurs Carry et Overflow.

Le contenu du registre A constitue le premier terme de la comparaison et la valeur lue en mémoire vive à l'adresse Y_2 en constitue le deuxième terme.

Eléments modifiés :

- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	(A) > y_2
0	1	(A) < y_2
1	0	(A) = y_2

Déroutements : standard

Exemples :

```
CMP     EPDL1
CMP     =3
CMP     #EPDC1
CMP     @EPDL2
CMP     @#EPDC20,x
CMP     @EPDL20,x
```

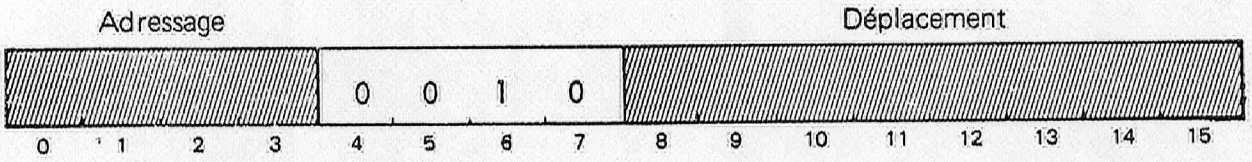
VII-6. INCREMENTATION ET DECREMENTATION DE REGISTRES

ICX	(Increment X)	Incrémentation de X
DCX	(Decrement X)	Décrémentation de X
ICL	(Increment L)	Incrémentation de L
DCL	(Decrement L)	Décrémentation de L

NOM : Incrémentation de X (InCrement X) **ICX**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	32	2,2
PX	E2	2,2
P	F2	2,2

Fonction : $(X) + y_2 \longrightarrow (X)$

La valeur lue en mémoire vive à l'adresse Y_2 est additionnée au contenu du registre X. Le résultat est rangé dans le registre X.

Eléments modifiés :

- Registres : X
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
1	-	Report
-	1	Débordement

Déroutements : standard

Exemples :

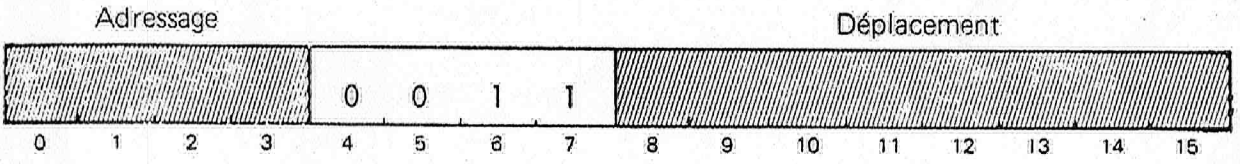
```
ICX     EPDL11
ICX     =&1A,x
ICX     =22
ICX     = 0,X     X multiplié par 2
```

DCX

NOM : Décrémentation de X (DeCrement X)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	33	2,2
PX	E3	2,2
P	F3	2,2

Fonction : $(X) - y_2 \longrightarrow (X)$

La valeur lue en mémoire vive à l'adresse Y_2 est soustraite du contenu du registre X. Le résultat est rangé dans le registre X.

Eléments modifiés :

- Registres : X
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
1	-	Report
-	1	Débordement

Déroutements : standard

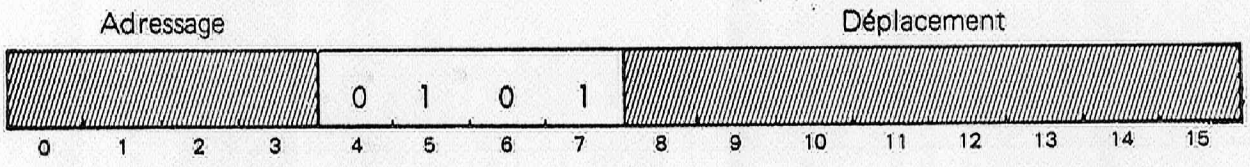
Exemples :

```
DCX     EPDL1
DCX     =3,x
DCX     =9
```

NOM : Incrementation de L (InCrement L) **ICL**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	35	2,2
PX	E5	2,2
P	F5	2,2

Fonction : $(L) + y_2 \longrightarrow (L)$

La valeur lue en mémoire vive à l'adresse Y_2 est additionnée au contenu du registre L. Le résultat est rangé dans le registre L.

Eléments modifiés :

- Registres : L

Déroutements : standard

Exemples :

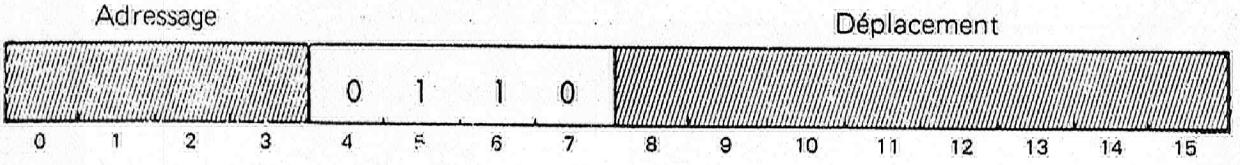
```
ICL     EPDL5
ICL     =50,x
ICL     =255
```

DCL

NOM : Decrementation de L (DeCrement L)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	36	2,2
PX	E6	2,2
P	F6	2,2

Fonction : $(L) - y_2 \longrightarrow (L)$

La valeur lue en mémoire vive à l'adresse Y_2 est soustraite du contenu du registre L. Le résultat est rangé dans le registre L.

Eléments modifiés :

- Registre : L

Déroutements : standard

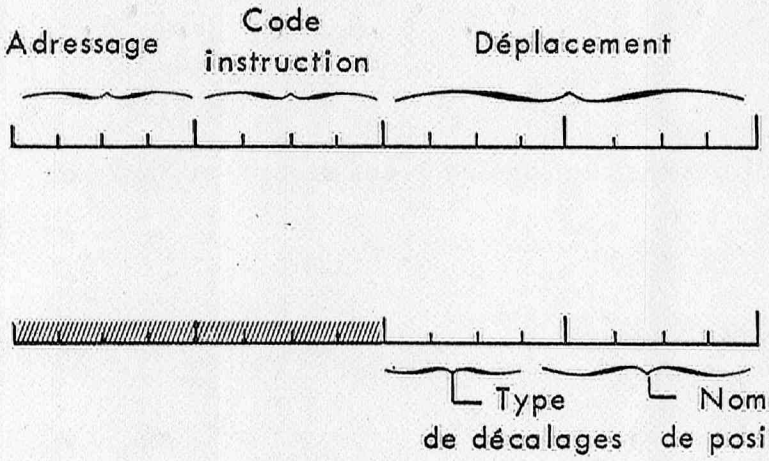
Exemples :

```
DCL     EPDL5
DCL     =5,x
DCL     =&1F
```

VII-7. DECALAGES

VII-7.1. Introduction

Il existe deux instructions de base pour les décalages : SHR (Shift register) et SHC (Shift Complementer). Ces deux instructions utilisent le mot situé à l'adresse Y_2 pour spécifier le type de décalage et le nombre de positions dont il faut décaler.



Il est bien évident qu'en mode paramètre, le plus fréquemment utilisé, il y a confusion entre les deux mots décrits ci-dessus.

Pour la commodité de la description, chacune des instructions dérivées de SHR et SHC sera décrite séparément.

Leurs mnémoniques sont reconnus par MITRAS II.

Remarque 1 :

Un cas particulier de SHC n'entre pas dans le groupe des décalages; il s'agit de DITR.

Notée pour mémoire dans l'instruction SHC, elle sera décrite en détail avec les branchements système.

Remarque 2 :

Le mode d'adressage "direct local" n'a de sens que pour SHR et SHC et non pour leurs dérivées (SLLS etc, SLLD etc.).

VII-7.2. Description

Les instructions de décalage sont décrites dans cet ordre :

	SHR	(Shift Register)	Décalage
Dérivées de SHR	SLLS	(Shift Left Logical Simple)	Décalage logique gauche de A
	SRCS	(Shift Right Circular Simple)	Décalage circulaire droit de A
	SAD	(Shift Arithmetic Double)	Décalage arithmétique droit de E, A
	SLCD	(Shift Left Circular Double)	Décalage circulaire gauche de E, A
	SLCS	(Shift Left Circular Simple)	Décalage circulaire gauche de A

Dérivées de SHR	SAS	(Shift Arithmetic Simple)	Décalage arithmétique droit de A
	SRLS	(Shift Right logical Simple)	Décalage logique droit de A
	SRCD	(Shift Right Circular Double)	Décalage circulaire droit de E, A
Dérivées de SHC	SHC	(Shift Complementar)	Décalage complémentaire
	SLLD	(Shift Left Logical Double)	Décalage logique gauche de E, A
	SRLD	(Shift Right Logical Double)	Décalage logique droit de E, A
	PTY	(Parity)	Calcul de parité
	NLZ	(Normalization)	Normalisation

Dans la description du code hexadécimal des instructions dérivées de SHR et SHC, on utilise la notation suivante :

nn — code hexadécimal de l'instruction SHR ou SHC

y-n — y : nombre de positions décalées ; n : code complémentaire exprimé en hexadécimal

Exemple SRLS = 5

Adressage	0	0	0	0	1	1	0	0	0	1	0	1
0 1 2 3	4	5	6	7	8	9	10	11	12	13	14	15

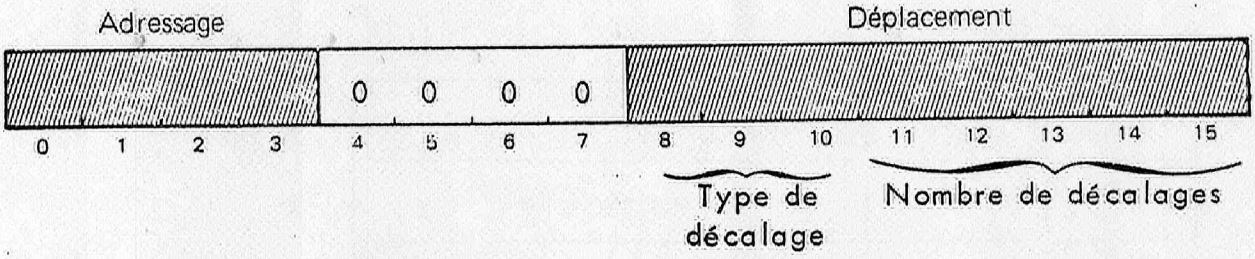
E0 C-n n = 5 E0 C5

SHR

NOM : Décalage (SHift Register)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal
DL	30
PX	E0
P	F0

Les temps d'exécution varient suivant le type de décalage.

Fonction : r décalé ⟶ r (r signifiant A ou E, A)

- décalage arithmétique : r_0 répété à chaque décalage de 1 position
- décalage circulaire : r_0 est considéré comme suivant r

N_{11-15} nombre de décalage ($0 \leq N_{11-15} \leq 31$)

N_{8-10} type de décalage

- 0 Logique gauche de A (SLLS)
- 1 Circulaire droit de A (SRCS)
- 2 Arithmétique droit de E, A (SAD)
- 3 Circulaire gauche de E, A (SLCD)
- 4 Circulaire gauche de A (SLCS)
- 5 Arithmétique droit de A (SAS)
- 6 Logique droit de A (SRLS)
- 7 Circulaire droit de E, A (SRCD)

Eléments modifiés :

- Registres : A (et E pour les décalages doubles)

- Positions mémoire :

- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de r
≠	≠	si décalage de zéro position

Déroutements : standard

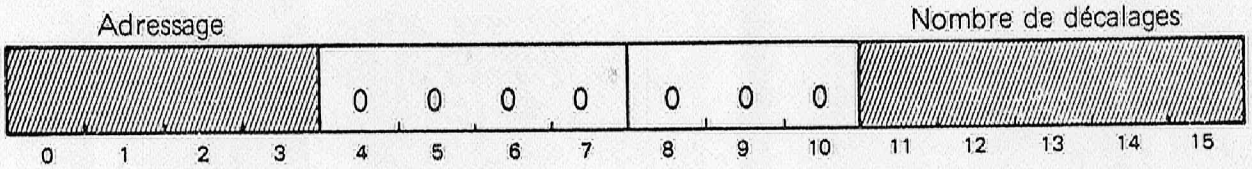
Exemples :

```
SHR     EPDL5 (équivalent à SRCD = 31)
SHR     =11,x (si (x) = 0, équivalent à SLLS = 11)
SHR     =&41  (équivalent à SAD = 1)
```

NOM : Décalage logique gauche de A (Shift Left Logical Simple) **SLLS**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
PX	E0 0-n	4,3 + 1,2 n
P	F0 0-n	4,3 + 1,2 n

= nombre de positions décalées

Fonction : (A) décalé $\longrightarrow$ (A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre A est décalé de n positions sur la gauche. Les bits de poids faibles sont complétés avec des 0.

Eléments modifiés :

- Registres : A
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de A
#	#	si décalage initial = 0

Déroutements : standard

Exemples :

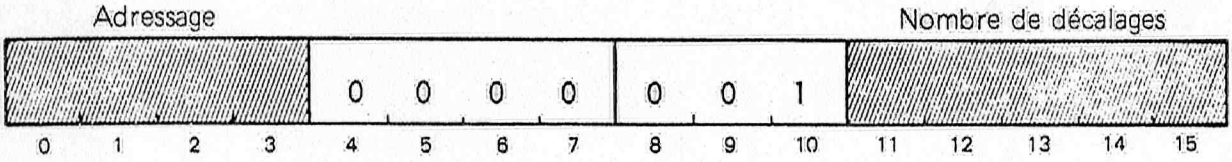
```
SLLS    =SPDL3,x
SLLS    =2
LDX     =2          } équivalent à SLLS = 6
SLLS    =4,x
```

SRCS

NOM : Décalage circulaire droit de A (Shift Right Circular Simple)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
PX	E0 2-n	4,3 + 1,5 n
P	E0 2-n	4,3 + 1,5 n

n = nombre de positions décalées

Fonction : (A) décalé circulairement --→ (A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre A est décalé circulairement de n positions sur la droite, le bit 0 faisant suite au bit 15.

Eléments modifiés :

- Registres : A
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de A
#	#	si décalage initial = 0

Déroutements : standard

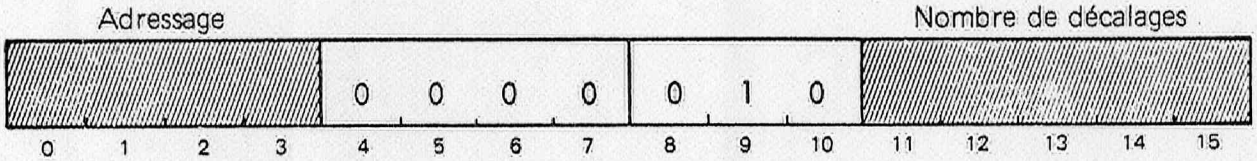
Exemples :

```
SRCS    =5,x
SRCS    =SPDL3
```

NOM : Décalage arithmétique droit de E et A (Shift Arithmetic Double) **SAD**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	E0	4-n	4,3 + 2,1 n
P	F0	4-n	4,3 + 2,1 n

n = nombre de positions décalées

Fonction : (E, A) décalé → (E, A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre étendu E, A est décalé arithmétiquement de n positions sur la droite. Le bit 0 (bit de signe) est reporté à chaque décalage.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de A
#	#	si décalage initial = 0

Déroutements : standard

Exemples :

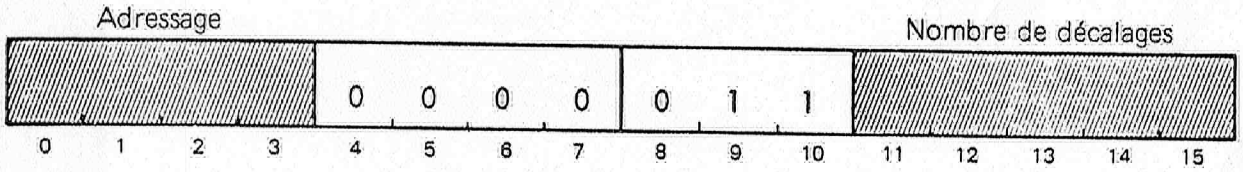
```
SAD     =SPDL2,x
SAD     =&03
```

SLCD

NOM : Décalage circulaire gauche de E et A (Shift Left Circular Double)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	E0	6-n	4,3 + 2,1 n
P	F0	6-n	4,3 + 2,1 n

n = nombre de positions décalées

Fonction : (E, A) décalé $\longrightarrow$ (E, A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre étendu E, A est décalé circulairement de n positions sur la gauche, le bit 31 faisant suite au bit 0.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de E
#	#	si décalage initial = 0

Déroutements : standard

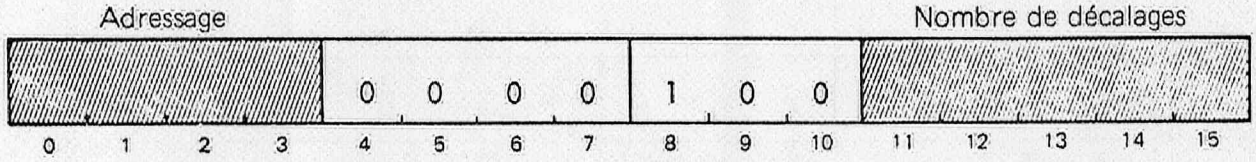
Exemples :

```
SLCD    =15,x
SLCD    =1
```

NOM : Décalage circulaire gauche de A (Shift Left Circular Simple) **SLCS**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	E0	8-n	4,3 + 1,5 n
P	F0	8-n	4,3 + 1,5 n

n = nombre de positions décalées

Fonction : (A) décalé $\longrightarrow$ (A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre A est décalé circulairement de n positions sur la gauche, le bit 15 faisant suite au bit 0.

Eléments modifiés :

- Registres : A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de A
#	#	si décalage initial = 0

Déroutements : standard

Exemples :

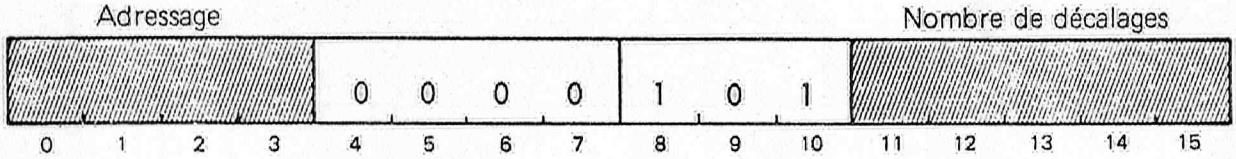
```
SLCS    =2,x
SLCS    =7
```

SAS

NOM : Décalage arithmétique droit de A (Shift Arithmetic Simple)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	E0	A-n	4,3 + 1,5 n
P	F0	A-n	4,3 + 1,5 n

n = nombre de positions décalées

Fonction : (A) décalé $\longrightarrow$ (A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre A est décalé arithmétiquement de n positions sur la droite. Le bit 0 (bit de signe) est reporté à chaque décalage.

Eléments modifiés :

- Registres : A
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de A
#	#	si décalage initial = 0

Déroutements : standard

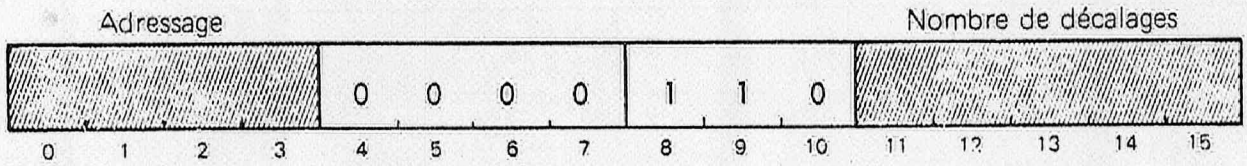
Exemples :

```
SAS     =&2,x
SAS     =SPDL2
```

NOM : Décalage logique droit de A (Shift Right Logical Simple) **SRLS**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	E0	C-n	4,3 + 1,5 n
P	F0	C-n	4,3 + 1,5 n

n = nombre de positions décalées

Fonction : (A) décalé ⟶ (A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre A est décalé de n positions sur la droite. Les bits de poids forts sont complétés avec des 0.

Eléments modifiés :

- Registres : A
- Indicateurs : C-O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de A
#	#	si décalage initial = 0

Déroutements : standard

Exemples :

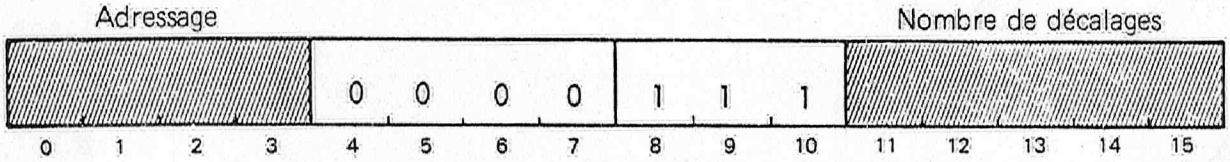
```
SRLS    =0,x
SRLS    =8
```

SRCD

NOM : Décalage circulaire droit de E et A (Shift Right Circular Double)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	E0	E-n	4,3 + 2,7 n
P	F0	E-n	4,3 + 2,7 n

n = nombre de positions décalées

Fonction : (E,A) décalé $\longrightarrow$(E,A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre étendu E, A est décalé circulairement de n positions sur la droite, le bit 0 faisant suite au bit 31.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de A
#	#	si décalage initial = 0

Déroutements : standard

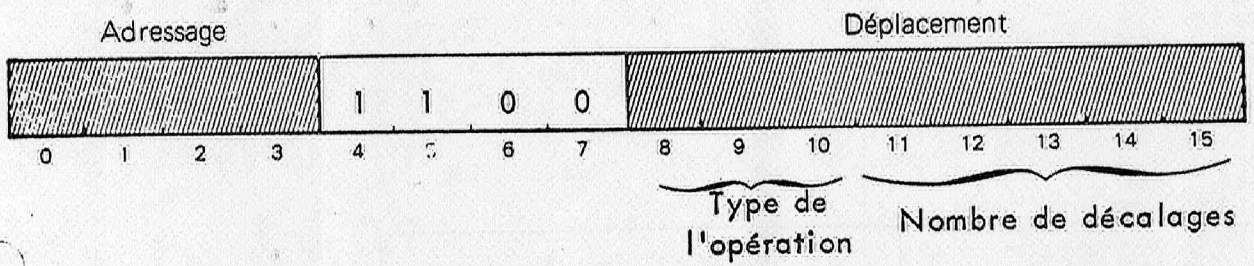
Exemples :

```
SRCD    =18,x
SRCD    =&12
```

NOM : Décalage complémentaire (SHift Complementar) **SHC**

Classe : 1 Privilégiée : non (sauf pour $N_{10} = 1$. Cas de DITR) Optionnelle : oui

Code de l'instruction :



Si $N_{10} = 1$, N_{11-15} est indifférent

Mode d'adressage	Code Hexadécimal
DL	3C
PX	EC
P	FC

Les temps d'exécution varient suivant le type de décalage.

Fonction : r décalé ⟶ r (r signifiant A ou E, A)

N_{11-15} nombre de décalage

$0 \leqslant N_{11-15} \leqslant 32$

N_{8-10} type de l'opération

- 0 Décalage logique gauche de E, A (SLLD)
  N_{11-15} = nombre de décalages
- 4 Décalage logique droit de E, A (SRLD)
  N_{11-15} = nombre de décalages
- 2 Calcul de parité sur A (PTY)
  N_{11-15} = nombre de décalages
  Après exécution, E = nombre de bit à 1 sorti de (A)
- 6 Normalisation de E, A (NLZ)
  N_{11-15} = nombre de décalage maximum

1
3 } Désactivation d'IT rapide (DITR)
5 } Seul le bit N_{10} est pris en compte lorsqu'il est à 1
7

Eléments modifiés :

- Registres : A (et E pour les décalages doubles)

- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	#	dernier bit sorti
#	#	si décalage de zéro position

Remarque :

Il s'agit ici d'une utilisation générale. Pour plus de détails se reporter à chacune des quatre instructions suivantes.

Déroutements : Instruction inexistante et standard
(Violation de mode pour DITR)

Divers : L'instruction DITR sera décrite séparement avec les instructions de commande.

Exemples :

SHC EPDL12 (équivalent à SRLD = 2)

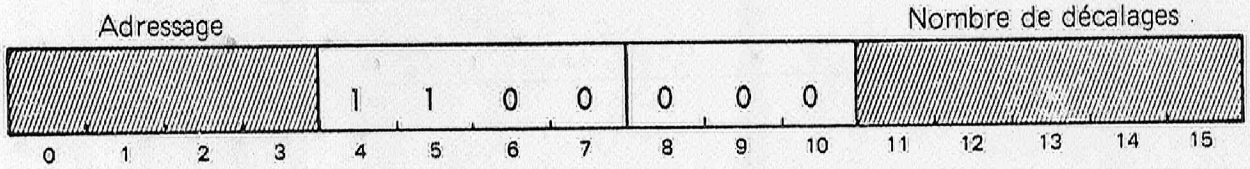
SHC =5,x si x = 5, équivalent à SLLS = 10)

SHC &48 (équivalent à PTY = 8)

NOM : Décalage logique gauche de E et A (Shift Left Logical Double) **SLLD**

Classe : 1 Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	EC	0-n	4,9 + 1,2 n
P	FC	0-n	4,9 + 1,2 n

= nombre de positions décalées

Fonction : (E, A) décalé → (E,A)
$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre étendu E, A est décalé de n positions sur la gauche. Les bits de poids faibles sont complétés par des 0.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	≠	dernier bit sorti de E
#	#	si décalage de zéro position

Déroutements : Instruction inexistante et standard

Exemples :

```
SLLD    =5,x
SLLD    =&0E
```

SRLD

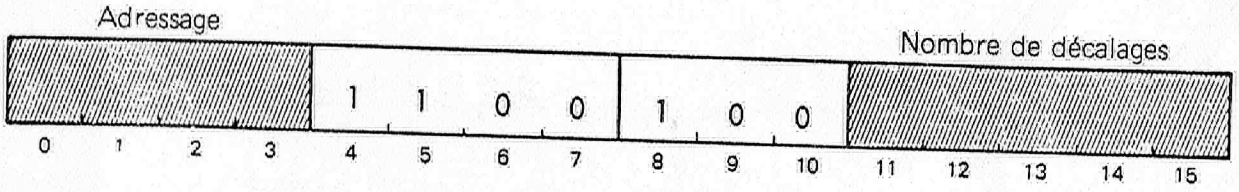
NOM : Décalage logique droit de E et A (Shift Right Logical Double)

Classe : 1

Privilégiée : non

Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	EC	8-n	4,9 + 1,8 n
P	FC	8-n	4,9 + 1,8 n

n = nombre de positions décalées

Fonction : (E, A) décalé $\longrightarrow$ (E, A)
$n = ((P))_{11-15}$

Le contenu du registre étendu E, A est décalé de n position sur la droite. Les bits de poids forts sont complétés par des 0.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	#	dernier bit sorti de A
#	#	si décalage initial = 0

Déroutements : Instruction inexistante et standard

Exemples :

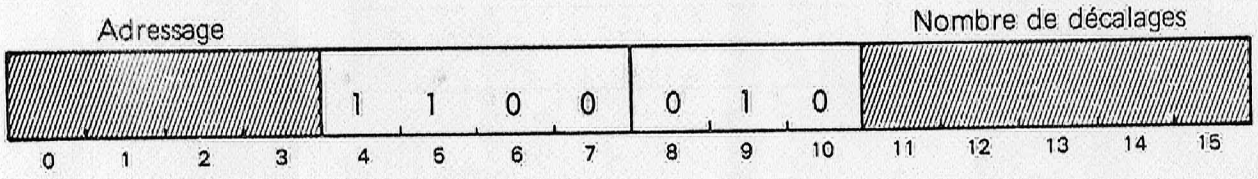
```
SRLD    =3,x
SRLD    =31
```

NOM : Calcul de parité (PariTY)

PTY

Classe : 1 Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	EC	4-n	5,8 + 1,2 n + 0,3 m
P	FC	4-n	5,8 + 1,2 n + 0,3 m

n = nombre de position décalées

m = nombre de bits à 1 rencontrés

Fonction : (A) décalé → (A)

(E) = nombre de bit à 1 sorti de (A)

$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre A est décalé circulairement de n positions sur la gauche. A la fin de l'instruction, le registre E contient le nombre de bits égaux à 1 qui sont sortis du registre A.

Eléments modifiés :

- Registres : A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0/1	-	dernier bit sorti de A
-	0/1	$(\overline{E_{15}})$ = bit de parité
#	#	si décalage de zéro position

Déroutements : Instruction inexistante et standard

Exemples :

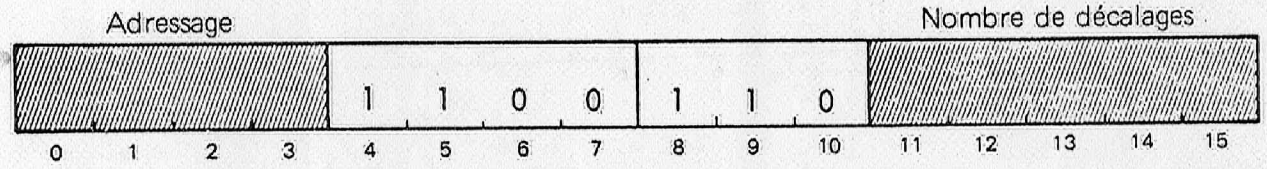
PTY =2,x

PTY =&F

NOM : Normalisation (NormaLiZation) **NLZ**

Classe : 1 Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	EC	C-n	4,9 + 1,8 n
P	FC	C-n	4,9 + 1,8 n

n = nombre de positions décalées

Fonction : (E, A) décalé → (E, A)

X décrémenté du nombre effectif de décalage

$n = ((P))_{11-15} = N_{11-15}$

Le contenu du registre étendu E, A est décalé à gauche jusqu'à ce que le bit 0 soit différent du bit 1 ou jusqu'à ce que l'on ait décalé de n positions. Le contenu du registre X est décrémenté du nombre effectif de positions décalées.

Eléments modifiés :

- Registres : E - A - X
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	Normalisation 01 .....
0	1	Arrêt par compte nul
1	0	Normalisation 10 .....

Déroutements : Instruction inexistante et standard

Exemples :

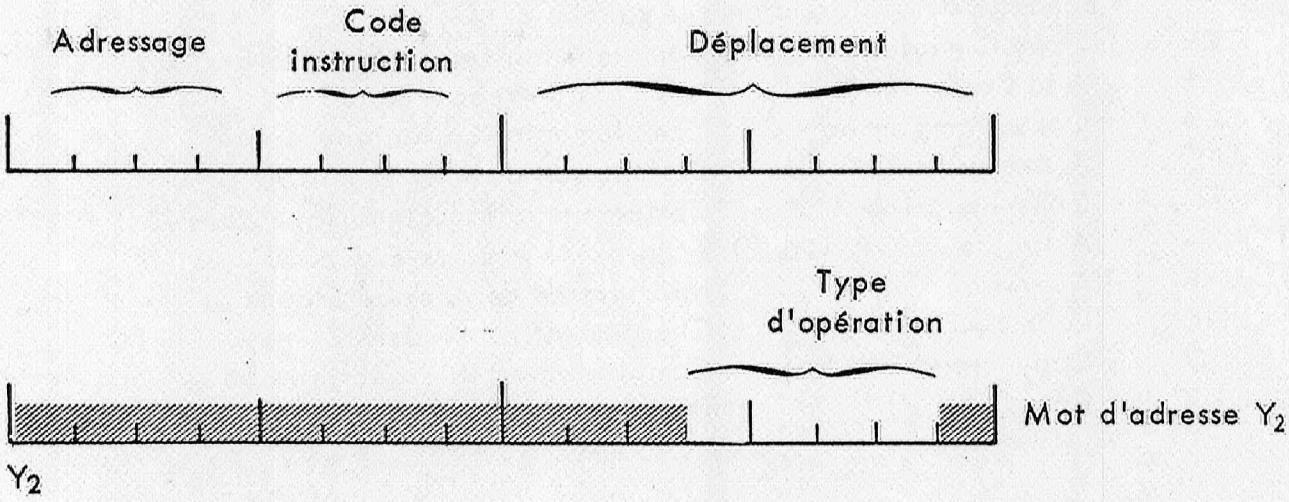
NLZ =10,x

NLZ =20

VII-8. OPERATIONS ENTRE REGISTRES

VII-8.1. Introduction

Il existe une instruction de base pour les opérations entre registres : SRG (Set Register). Cette instruction utilise le mot situé à l'adresse Y_2 pour spécifier le type d'opération effectuée.



Il est bien évident qu'en mode paramètre, normalement le seul utilisé, il y a confusion entre les deux mots décrits ci-dessus.

Pour la commodité de la description, chacune des instructions dérivées de SRG sera décrite séparément.

Leurs mnémoniques sont reconnus par MITRAS I et MITRAS II

Remarque 1 :

Deux cas particuliers de SRG n'entrent pas dans le groupe des opérations sur registres; il s'agit de RTS et RSV. Notées pour mémoire dans l'instruction SRG, elles seront décrites en détail avec les branchements système.

Remarque 2 :

Tout autre mode d'adressage que "paramètre" n'a pas de sens avec les instructions dérivées de SRG.

VII-8.2. Description

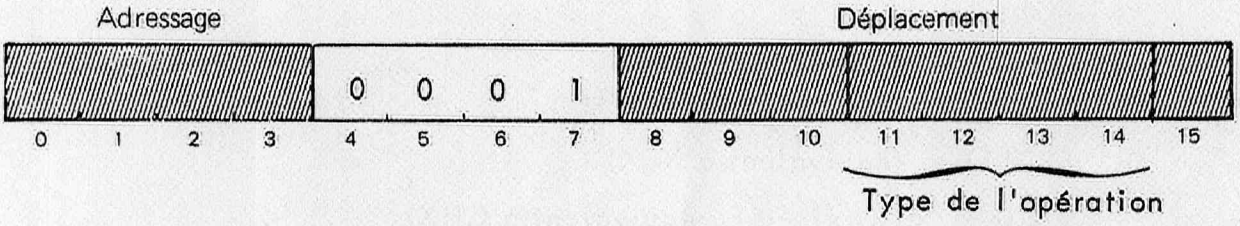
Les opérations entre registres seront décrites dans cet ordre :

	SRG	(Set Register)	Operations entre registres
	XAE	(Exchange A and E)	Echange de A et E
	XAX	(Exchange A and X)	Echange de A et X
	XEX	(Exchange E and X)	Echange de E et X
	XAA	(Exchange left byte of A and righ byte of A) Echange des octets droit et gauche de A	
Dérivées de SRG	CCE	(Copy Complement E)	Complémentation logique de E
	ACE	(Add Carry and E)	Addition du report dans E
	CCA	(Copy Complement A)	Complementation logique de A
	AEE	(A exclusive OR with E)	Disjonction de A et E dans A
	CNX	(Copy negative X)	Complémentation algébrique dans X
	AIE	(A Inclusive OR with E)	Réunion de A et de E dans A
	AAE	(A And E)	Intersection de A et de E dans A
	LNE	(Load Negative E)	Chargement de -1 dans E
	CNA	(Copy Negative A)	Complémentation algébrique de A
	CHX	(Copy Half X)	Division de X par deux

NOM : Opérations entre registres (Set ReGister) **SRG**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal
DL	31
PX	E1
P	F1

Les temps d'éxécution varient suivant le type d'opération effectuée

Fonction : N_{11-14}

- 0 Retour section → (RTS)
- 1 Echange de (A) et (E) → (XAE)
- 2 Echange de (A) et (X) → (XAX)
- 3 Echange de (E) et (X) → (XEX)
- 4 Echange de (A_{0-7}) et (A_{8-15}) → (XAA)
- 5 Complémentation logique de (E) → (CCE)
- 6 Retour superviseur → (RSV)
- 7 Addition du report dans (E) → (ACE)
- 8 Complémentation logique de (A) → (CCA)
- 9 Disjonction de (A) et de (E) dans (A) → (AEE)
- A Complémentation algébrique dans (X) → (CNX)
- B Réunion de (A) et de (E) dans (A) → (AIE)
- C Intersection de (A) et de (E) dans (A) → (AAE)
- D Chargement de - 1 dans (E) → (LNE)
- E Complémentation algébrique dans (A) → (CNA)
- F Division de X par deux → (CHX)

Eléments modifiés :

- Registres : Se reporter à chaque instruction

- Indicateurs : Se reporter à chaque instruction dérivée

Déroutements : standard

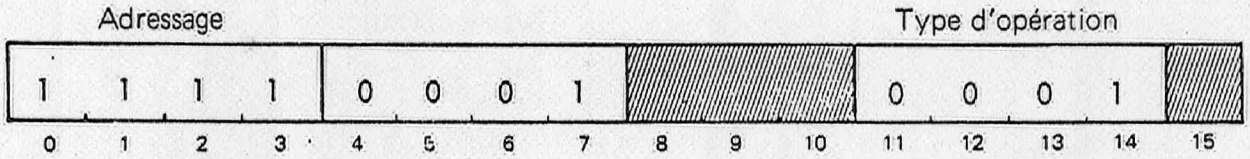
Exemples :

SRG	EPDL11	(équivalent à XEX)
SRG	=&10,x	(si (x) = &E, équivalent à CHX)
SRG	=14	(équivalent à ACE)

NOM : Echange de A et E (eXchange A and E) **XAE**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	02	4,3

Fonction : (A) ⟷ (E)

Le contenu du registre A et le contenu du registre E sont échangés

Eléments modifiés :

- Registres : E - A

Déroutements : standard

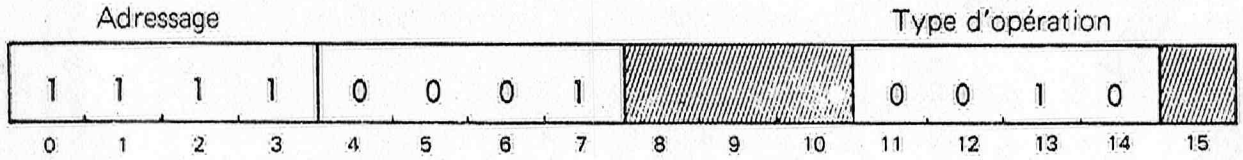
Exemples : XAE

XAX

NOM : Echange de A et de X (eXchange A and X)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	04	4,3

Fonction : (A) ⟷ (X)

Le contenu du registre A et le contenu du registre X sont échangés.

Eléments modifiés :

- Registres : A - X

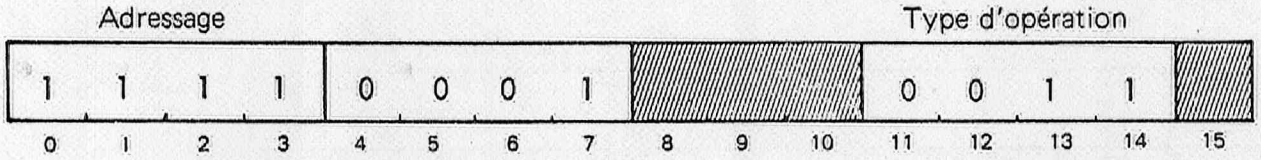
Déroutements : standard

Exemples : XAX

NOM : Echange de E et de X (eXchange E and X) **XEX**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en μs
P	F1	06	4,3

Fonction : (E) ⟷ (X)

Le contenu du registre E et le contenu du registre X sont échangés

Eléments modifiés :

- Registres : E - X

Déroutements : standard

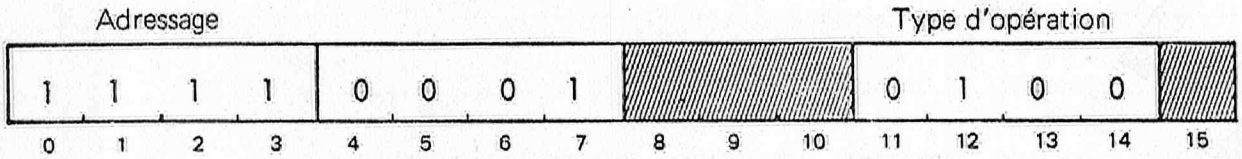
Exemples : XEX

XAA

NOM : Echange des octets droit et gauche de A
(eXchange left byte of A and right byte of A)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
P	F1 08	2,8

Fonction : $(A_{0-7}) \longleftrightarrow (A_{8-15})$

L'octet de poids forts du registre A et l'octet de poids faibles du registre A sont échangés.

Eléments modifiés :

- Registres : A

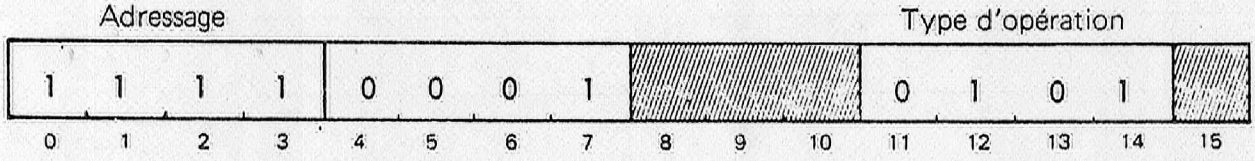
Déroutements : standard

Exemples : XAA

NOM : Complementation logique de E (Copy Complement E) **CCE**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	0A	2,8

Fonction : $(\overline{E}) \longrightarrow (E)$

Chaque bit dû registre E à 0 ou à 1 est remplacé par son complément, respectivement 1 ou 0.

Eléments modifiés :

- Registres : E

Déroutements : standard

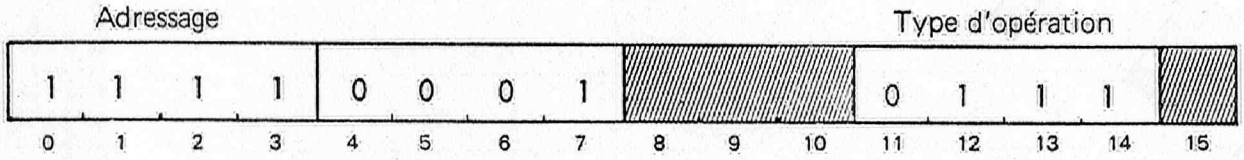
Exemples : CCE

ACE

NOM : Addition du report dans E (Add Carry and E)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	0E	3,4

Fonction : (E) + C ⟶ (E)

La valeur de l'indicateur Carry est additionnée au contenu du registre E. Le résultat est rangé dans le registre E.

Eléments modifiés :

- Registres : E
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
1	-	Report
-	1	Débordement

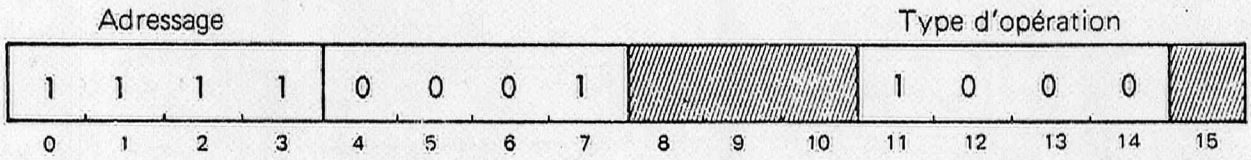
Déroutements : standard

Exemples : ACE

NOM : Complementation logique de A (Copy Complement) **CCA**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	10	2,8

Fonction : $\overline{(A)} \longrightarrow (A)$

Chaque bit du registre A à 0 ou à 1 est remplacé par son complément, respectivement 1 ou 0.

Eléments modifiés :

- Registres : A

Déroutements : standard

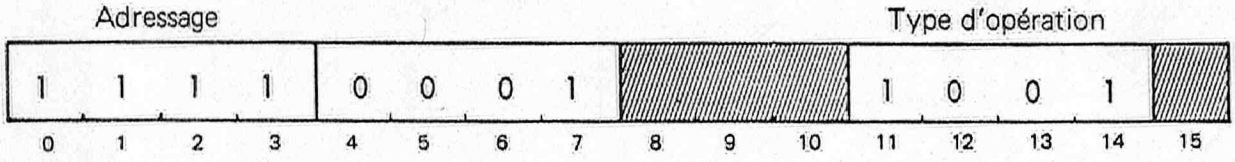
Exemples : CCA

AEE

NOM : Disjonction de A et de E dans A (A Exclusive or with E)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	12	3,1

Fonction : (A) ⊕ (E) → (A)

Disjonction (OU exclusif) entre le contenu du registre A et le contenu du registre E. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	A > 0
0	1	A < 0
1	0	A = 0

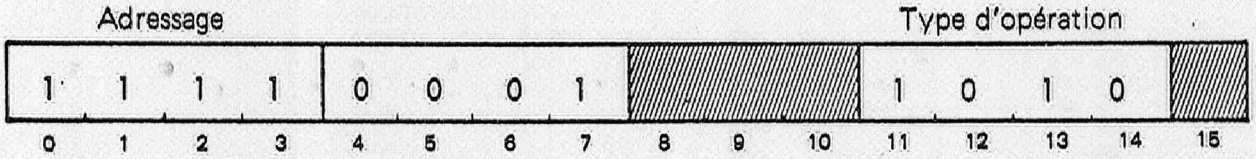
Déroutements : standard

Exemples : AAE

NOM : Complémentation algébrique dans X (Copy Negative X) **CNX**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	14	3,1

Fonction : - (X) ⟶ (X)

Complémentation algébrique (complément à 2^{16}) du contenu du registre X.

Le résultat est rangé dans le registre X.

Eléments modifiés :

- Registres : X

Déroutements : standard

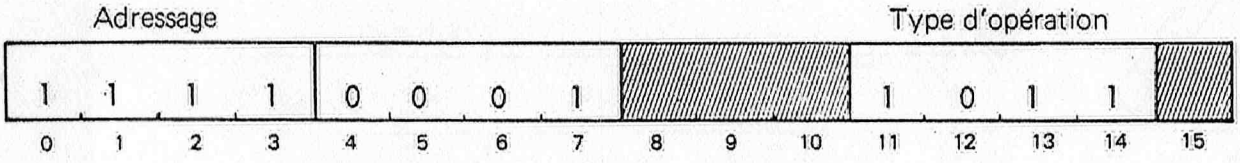
Exemples : CNX

AIE

NOM : Réunion de A et de E dans A (A Inclusive or with E)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	16	3,1

Fonction : (A) ∨ (E) → (A)

Réunion (OU inclusif) entre le contenu du registre A et le contenu du registre E. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	A > 0
0	1	A < 0
1	0	A = 0

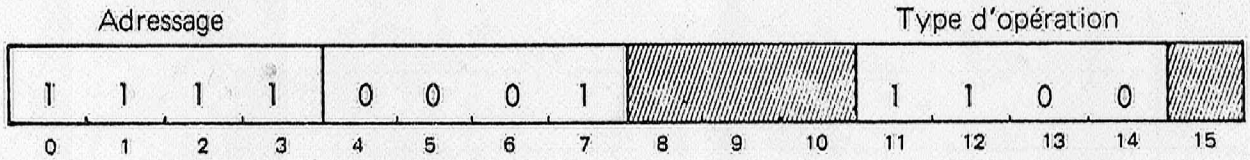
Déroutements : standard

Exemples : AIE

NOM : Intersection de A et de E dans A (A And E) **AAE**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	18	3,1

Fonction : (A) ∧ (E) ⟶ (A)

Intersection (ET) entre le contenu du registre A et le contenu du registre E. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	A > 0
0	1	A < 0
1	0	A = 0

Déroutements : standard

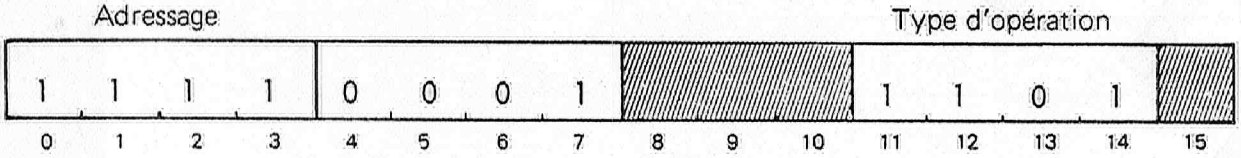
Exemples : AAE

LNE

NOM : Chargement de - 1 dans E (Load Negative E)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	1A	3,1

Fonction : - 1 ⟶ (E)

La valeur - 1 est chargé dans le registre E.

Eléments modifiés :

- Registres : E

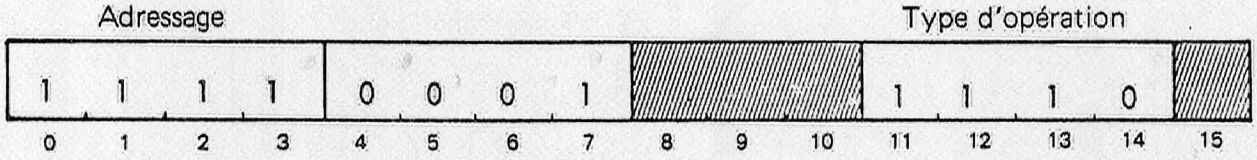
Déroutements : standard

Exemples : LNE

NOM : Complémentation algébrique dans A (Copy Negative A) **CNA**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	1C	3,4

Fonction : - (A) → (A)

Complémentation algébrique (complément à 2^{16}) du contenu du registre A. Le résultat est rangé dans le registre A.

Eléments modifiés :

- Registres : A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
1	-	Report
-	1	Débordement

Déroutements : standard

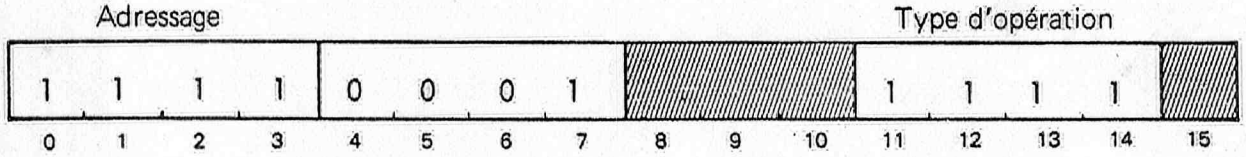
Exemples : CNA

CHX

NOM : Division de X par deux (Copy Half X)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
P	F1	1E	3,1

Fonction : (X) / 2 ⟶ (X)

Le contenu du registre X est décalé de une position sur la droite. Le bit de signe (bit 0) est reporté. Ceci permet d'obtenir la division du contenu du registre X par deux.

Eléments modifiés :

- Registres : X

Déroutements : standard

Exemples : CHX

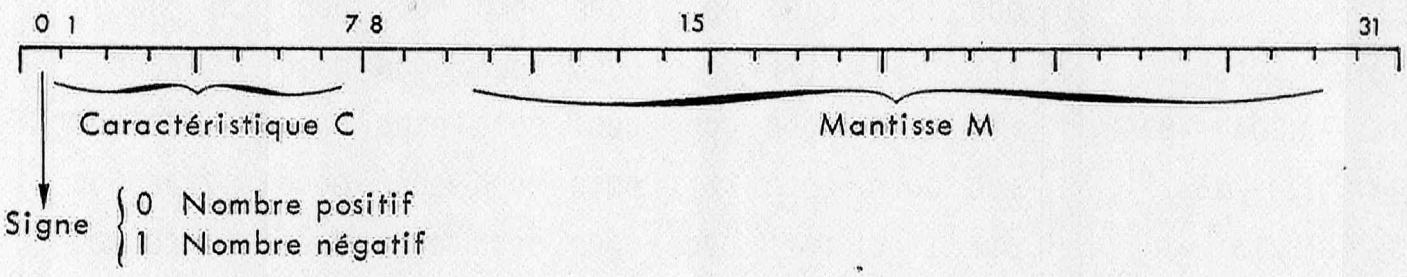
VII-9. ARITHMETIQUE VIRGULE FLOTTANTE

VII-9.1. Introduction

Ces instructions traitent des opérandes au format virgule flottante simple et sont réalisées par le dispositif optionnel "opérateur flottant".

Le format virgule flottante simple permet de représenter des nombres décimaux sur un double mot. Ce double mot a la structure suivante :

- Un signe (position zéro)
- Un exposant de base 16 dont la valeur augmentée de 64 est appelée caractéristique (positions 1 à 7).
- Une mantisse de 6 chiffres hexadécimaux (positions 8 à 31).



La définition formelle d'un nombre N exprimé en virgule flottante simple est la suivante :

- Si $N \geq 0$ $\quad N = M \times 16^{C-64}$

avec $M = 0$ ou $16^{-6} \leq M < 1$ et $0 \leq C \leq 127$

- Un nombre positif exprimé en virgule flottante ayant une mantisse nulle doit avoir une caractéristique nulle. C'est le nombre zéro.
- Un nombre flottant positif est dit normalisé si sa mantisse vérifie les inégalités : $1/16 \leq M < 1$.
- Un nombre négatif est représenté par le complément à 2 de sa valeur absolue. Plus précisément, la représentation de la valeur absolue d'un nombre négatif N comprend un signe (0), une caractéristique et une mantisse, l'ensemble formant un double mot. C'est le complément à 2 de ce double mot qui constitue la représentation de N.

Les instructions de l'arithmétique virgule flottante traitent normalement des opérandes normalisés (elles n'effectuent pas de pré-normalisation) et fournissent dans ce cas des résultats normalisés.

Si les opérandes ne sont pas normalisés, les résultats seront ou non normalisés suivant les cas.

Exemple de représentation en format flottant :

Nombre Décimal	FORMAT VIRGULE FLOTTANTE SIMPLE			Valeur Héxadécimale
	±	C	M	
$+(16^{+63})(1-2^{-24})$	0	111 1111	1111 1111 1111 1111 1111 1111	7F FFFFFF
$+(16^{+3})(5/16)$	0	100 0011	0101 0000 0000 0000 0000 0000	43 500000
$+(16^{-3})(209/256)$	0	011 1101	1101 0001 0000 0000 0000 0000	3D D10000
$+(16^{-63})(2047/4096)$	0	000 0001	0111 1111 1111 0000 0000 0000	01 7FF000
$+(16^{-64})(1/16)$	0	000 0000	0001 0000 0000 0000 0000 0000	00 100000
0	0	000 0000	0000 0000 0000 0000 0000 0000	00 000000
$-(16^{-64})(1/16)$	1	111 1111	1111 0000 0000 0000 0000 0000	FF F00000
$-(16^{-63})(2047/4096)$	1	111 1110	1000 0000 0001 0000 0000 0000	FF 801000
$-(16^{-3})(209/256)$	1	100 0010	0010 1111 0000 0000 0000 0000	C2 2F0000
$-(16^{+3})(5/16)$	1	011 1100	1011 0000 0000 0000 0000 0000	BC B00000
$-(16^{+63})(1-2^{-24})$	1	000 0000	0000 0000 0000 0000 0000 0001	80 000001

Limites du format flottant normalisé :

$0,86362 \times 10^{-76} < N < 0,72370 \times 10^{+76}$

VII-9.2. Description

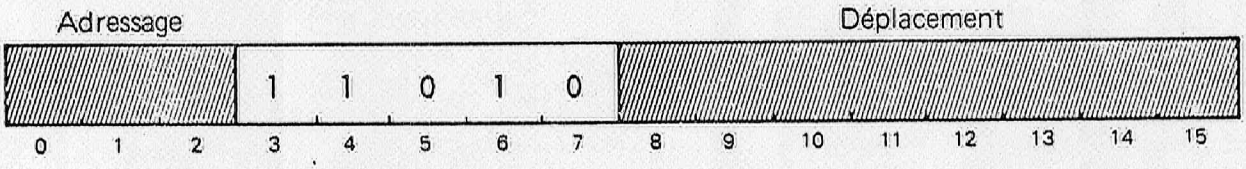
Les instruction traitant l'arithmétique virgule flottante permettent l'exécution des quatre opérations et sont décrites dans la suite dans l'ordre suivant :

FAD	(Floating Addition)	Addition de nombres flottants
FSU	(Floating Substraction)	Soustraction de nombres flottants
FMU	(Floating Multiplication)	Multiplication de nombres flottants
FDV	(Floating Division)	Division de nombres flottants

NOM : Addition de nombre flottants (Floating ADdition) **FAD**

Classe : 0' Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal
DL	1A
DG	5A
IL	7A
IGX	9A
ILX	BA

Fonction : $(E,A) + (Y_2, Y_2 + 2) \longrightarrow (E,A)$

Le nombre flottant contenu dans le registre étendu E, A est additionné avec le nombre flottant contenu dans le double mot d'adresse Y_2 en mémoire vive. Le résultat est rangé dans le registre étendu E, A.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	Résultat # 0, non débordé
0	1	Débordement supérieur
1	0	Résultat = 0
1	1	Débordement inférieur

Déroutements : standard

Exemples :

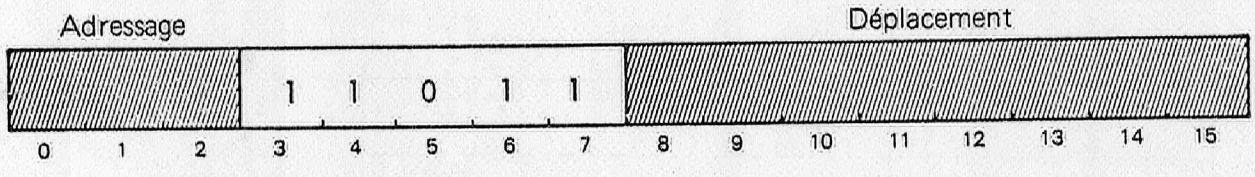
```
FAD     EPDL26
FAD     #EPDC26
FAD     @EPDL27
FAD     @#EPDC27,x
FAD     @EPDL27,x
```

NOM : Soustraction flottante (Floating SUBstraction)

FSU

Classe : 0' Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal
DL	1B
DG	5B
IL	7B
IGX	9B
ILX	BB

Fonction : $(E, A) - (Y_2, Y_2 + 2) \longrightarrow (E, A)$

Le nombre flottant contenu dans le double mot d'adresse Y_2 en mémoire vive est soustrait du nombre flottant contenu dans le registre E,A. Le résultat est rangé dans le registre étendu E, A.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	Résultat ≠ 0, non débordé
0	1	Débordement supérieur
1	0	Résultat = 0
1	1	Débordement inférieur

Déroutements : standard

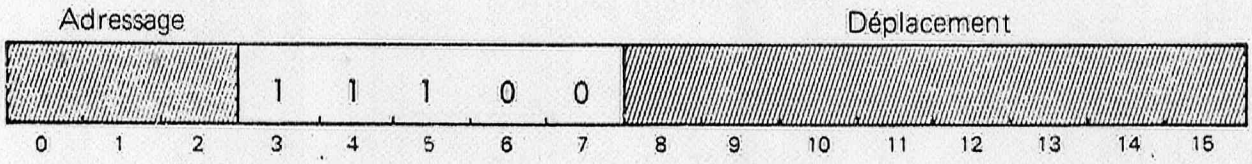
Exemples :

```
FSU      EPDL26
FSU      #EPDC26
FSU      @EPDL27
FSU      @#EPDC27,x
FSU      @EPDL27,x
```

NOM : Multiplication flottante (Floating MUltiplication) **FMU**

Classe : 0' Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal
DL	1C
DG	5C
IL	7C
IGX	9C
ILX	BC

Fonction : $(E, A) \times (Y_2, Y_2 + 2) \rightarrow E, A$

Le nombre flottant contenu dans le registre étendu E, A est multiplié par le nombre flottant contenu dans le double mot d'adresse Y_2 en mémoire vive. Le résultat est rangé dans le registre étendu E, A.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	Résultat ≠ 0, non débordé
0	1	Débordement supérieur
1	0	Résultat = 0
1	1	Débordement inférieur

Déroutements : standard

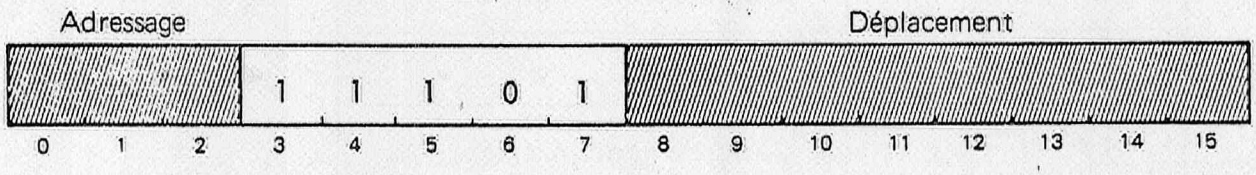
Exemples :

```
FMU     EPDL26
FMU     #EPDC26
FMU     @EPDL27
FMU     @#EPDC27,x
FMU     @EPDL27,x
```

NOM : Division flottante (Floating DiVision) **FDV**

Classe : 0' Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal
DL	1D
DG	5D
IL	7D
IGX	9D
ILX	BD

Fonction : (E, A) : (Y_2, Y_2 + 2) → (E, A)

Le nombre flottant contenu dans le registre étendu E, A est divisé par le nombre flottant contenu dans le double mot d'adresse Y_2 en mémoire vive. Le résultat est rangé dans le registre étendu E, A.

Eléments modifiés :

- Registres : E - A
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	Résultat ≠ 0, non débordé
0	1	Débordement supérieur
1	0	Résultat = 0
1	1	Débordement inférieur

Déroutements : standard

Exemples :

```
FDV     EPDL26
FDV     #EPDC26
FDV     @EPDL27
FDV     @#EPDC27,x
FDV     @EPDL27,x
```

VII-10. TRAITEMENT DES CHAINES

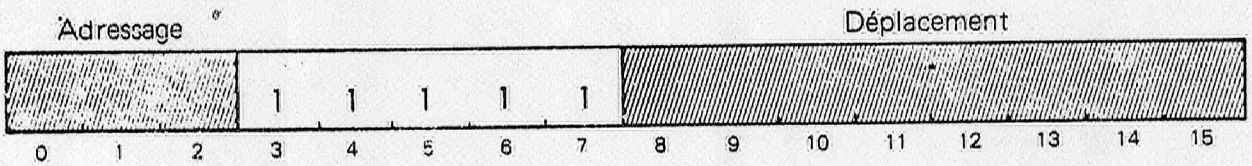
Les opérations sur les chaînes d'octet sont réalisées par les trois instructions optionnelles suivantes :

MVS	(Move byte String)	Déplacement d'une chaîne d'octets
CPS	(Compare String)	Comparaison d'un octet à une chaîne
TRS	(Translate String)	Transcodage d'une chaîne d'octets

NOM : Déplacement d'une chaîne d'octet (MoVe byte String) **MVS**

Classe : 0' Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	1F	3 + 3,3 n
DG	5F	3 + 3,3 n
IL	7F	4 + 3,3 n
IGX	9F	4 + 3,3 n
ILX	BF	4 + 3,3 n

n = nombre d'octets déplacés

Fonction : Pour α variant de (E) - 1 à 0, octet par octet

((G) + (A) + α) $\longrightarrow$ (Y + α)

En fin de transfert - 1 $\longrightarrow$ E

La chaîne d'octets, dont l'adresse de début par rapport à la base G est contenue dans le registre A et la longueur en octets dans le registre E, est rangée en mémoire vive à partir de l'adresse Y.

En fin de transfert le registre E contient -1 et le registre A reste inchangé.

Eléments modifiés :

- Registres : E

- Positions mémoire : (Y) à (Y + (E) - 1)

Déroutements : Instruction inexistante et standard

Divers : Cette instruction est interruptible entre chaque octet

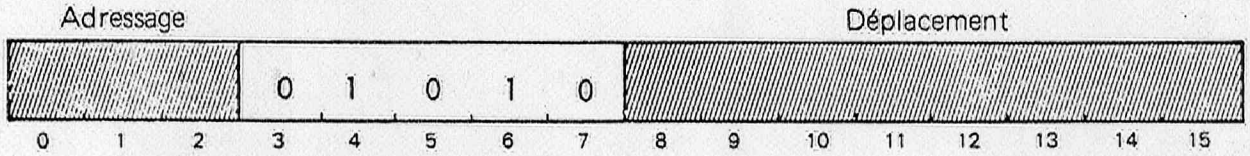
Exemples :

```
MVS     EPDL4
MVS     #EPDC4
MVS     @EPDL3
MVS     @#EPDC3,x
MVS     @EPDL3,x
```

NOM : Comparaison d'un octet à une chaîne (ComPare String) **CPS**

Classe : 0 Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	0A	4,2 + 3,4 n
P	2A	4,6 + 3,4 n
DG	4A	4,2 + 3,4 n
IL	6A	5,2 + 3,4 n
IGX	8A	5,2 + 3,4 n
ILX	AA	5,2 + 3,4 n

n = nombre de comparaisons faites

Fonction : Pour α variant de 0 à (E)-1 octet par octet.

$y = ((G) + (A) + \alpha)$

Après exécution :

- si trouvé : A = adresse du caractère dans la chaîne
- si non trouvé : A = adresse du premier caractère non traité

L'octet y lu en mémoire vive à l'adresse Y est comparé successivement aux octets de la chaîne dont l'adresse de début par rapport à la base G est contenue dans le registre A et la longueur dans le registre E.

Après exécution :

Si le caractère y appartient à la chaîne, le registre A contient l'adresse relative à la base G, du caractère trouvé et le registre E contient la longueur de la chaîne non traitée.

Si le caractère y n'appartient pas à la chaîne, le registre A contient l'adresse relative à la base G, du premier caractère non traité et le registre E contient la valeur zéro.

Eléments modifiés :

- Registres : E - A

- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	Caractère trouvé
0	1	Caractère non trouvé

Déroutements : Instruction inexistante et standard

Divers : Cette instruction est interruptible entre chaque comparaison d'octet.

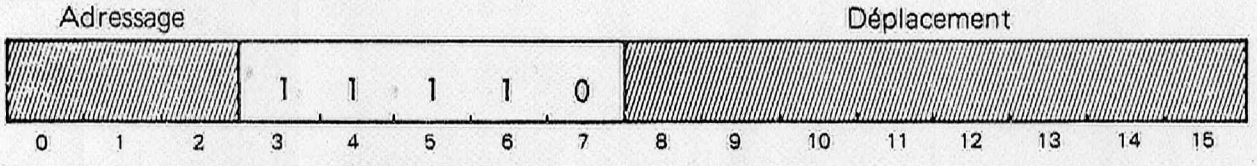
Exemples :

```
CPS         EPDL18
CPS         =&6C
CPS         #EPDC18
CPS         @EPDL19
CPS         @#EPDC19,x
CPS         @EPDL19,x
```

NOM : Transcodage d'une chaîne d'octets (TRanslate String) **TRS**

Classe : 0' Privilégiée : non Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	1E	4,5 + 2,75 n
DG	5E	4,5 + 2,75 n
IL	7E	5,5 + 2,75 n
IGX	9E	5,5 + 2,75 n
ILX	BE	5,5 + 2,75 n

n : nombre d'octets à transcoder

Fonction : Pour α variant de 0 à (E) - 1, octet par octet

$(Y + ((A) + (G) + \alpha)_e) \longrightarrow ((A) + (G) + \alpha)$

En fin de transfert $0 \longrightarrow (E)$

Soit un code O (origine) et un code R (résultat) ainsi qu'une table de transcodage de 256 octets consécutifs située à l'adresse calculée Y et constitué comme suit :

Adresse relative au début de la table (en hexadécimal)	Contenu
00	Valeur dans le code R correspondant à la valeur 00 dans le code O.
01	Valeur dans le code R correspondant à la valeur 01 dans le code O.

Adresse relative au début de la table (en hexadécimal)	Contenu
02	Valeur dans le code R correspondant à la valeur 02 dans le code O.
⋮	⋮
FF	Valeur dans le code R correspondant à la valeur FF dans le code O.

Le chaîne dont l'adresse relative à G est située dans A et la longueur dans E est convertie octet par octet à l'aide de la table de transcodage adressée par l'instruction TRS.

(Adresse calculée y), la chaîne résultat recouvrant octet par octet la chaîne origine.

La constitution de la table de transcodage est évidemment à la charge de l'utilisateur.

Eléments modifiés :

- Registres : A (si E initial ≠ 0) - E

- Positions mémoire : ((A)) à ((A) + (E) - 1)

Déroutements : Instruction inexistante et standard

Divers : Cette instruction est interruptible entre chaque mot.

Exemples :

```
TRS     EPDL18
TRS     #EPDC18
TRS     @EPDL19
TRS     @#EPDC19,x
TRS     @EPDL19,x
```

VII-11. BRANCHEMENTS

Les instructions de branchement servent normalement à interrompre la suite séquentielle des instructions dans un segment de programme.

Elles sont les suivantes :

BRU	(Branch Unconditional)	Branchement inconditionnel
BRX	(Branch with Index)	Branchement indexé
BCT	(Branch if Carry True)	Branchement si Carry est vrai
BOT	(Branch if Overflow True)	Branchement si Overflow est vrai
BCF	(Branch if Carry False)	Branchement si Carry est faux
BOF	(Branch if Overflow False)	Branchement si Overflow est faux
BAZ	(Branch if A equal to Zero)	Branchement si A est nul
BAN	(Branch if A Negative)	Branchement si A est négatif
BE	(Branch if Equal to)	Branchement si égal
BZ	(Branch if Equal to Zero)	Branchement si nul
BL	(Branch if Less Than)	Branchement si inférieur
BLZ	(Branch if Less Than Zero)	Branchement si négatif
BNE	(Branch on Not equal to)	Branchement si différent
BNZ	(Branch if Not equal to Zero)	Branchement si non nul
BGE	(Branch if greater than or Equal to)	Branchement si supérieur ou égal
BPZ	(Branch if Positive or equal to Zero)	Branchement si positif ou nul

Remarque 1 :

On dit qu'un indicateur est "vrai" s'il est positionné à 1 sinon on dit qu'il est "faux"

Remarque 2 :

Si le registre P a une valeur paire, l'instruction exécutée est à l'adresse absolue (P).

Si le registre P a une valeur impaire, l'instruction exécutée est à l'adresse absolue (P) - 1.

Remarque 3 :

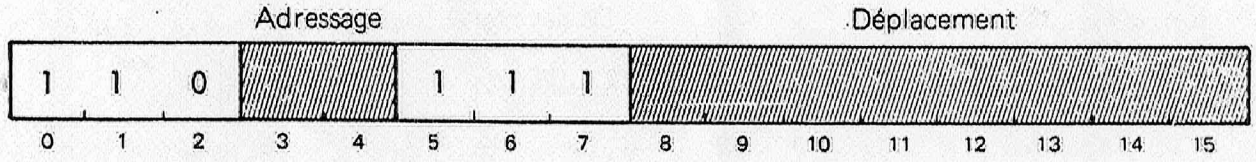
Plusieurs des branchements conditionnels correspondent à la même instruction de base. Les mnémoniques supplémentaires (non reconnus par MITRAS 1) sont destinés à faciliter la programmation suivant qu'ils sont utilisés après chargement ou après comparaison.

Instruction de base	Mnémoniques pour utilisation après chargement	Mnémoniques pour utilisation après comparaison
BCT	BZ	BE
BOT	BLZ	BL
BCF	BNZ	BNE
BOF	BPZ	BGE

NOM : Branchement inconditionnel (BRanch Unconditional) **BRU**

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
RP	C7	1,8
RM	CF	1,8
IL	D7	3
IG	DF	3

Fonction : Y → (P)

L'adresse Y est chargée dans le registre P ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Eléments modifiés :

- Registres : P

Déroutements : standard

Exemples :

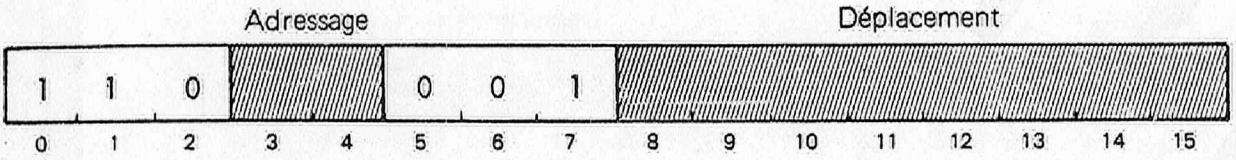
```
          BRU    @EPDL14
EPPL2     BRU    EPPL3+1
EPPL3     BRU    @#EPDC13
          BRU    EPPL3-1 → Equivalent à BRU  EPPL2
          BRU    $
```

BRX

NOM : Branchement indexé (BRanch with indeX)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
RP	C1	2,1
RM	C9	2,1
IL	D1	3,2
IG	D9	3,2

Fonction : RP : (P) + 2 De + 2 (X) → (P)

RM : (P) - 2 De - 2 (X) → (P)

IL : (De + (L) + (X)) + G' → (P)

IG : (De + (G) + (X)) + G' → (P)

L'adresse calculée Y est chargée dans le registre P ce qui provoque la poursuite de l'exécution à partir de l'adresse Y. En mode indirect, l'indexation est une pré-indexation, c'est-à-dire que l'indexation est exécuté avant l'indirection.

Le mode de calcul de l'adresse est indiqué ci-dessus.

Remarque :

Bien que BRX puisse s'utiliser en mode relatif plus, ou relatif moins, son usage normal est en mode indirect pour effectuer des branchements multiples en utilisant une table d'adresses.

Eléments modifiés :

- Registres : P

Déroutements : standard

Exemples :

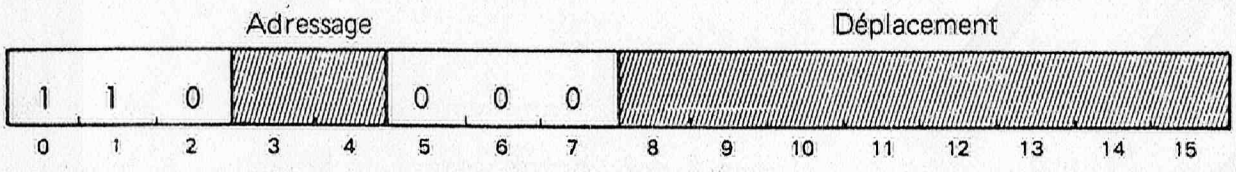
```
EPPL0   BRX     EPPL1       (pour X = 2, branchement en EPPL3)
EPPL1   BRX     EPPL3       (pour X = 1, branchement en EPPL4)
EPPL2   BRX     EPPL3       (pour X = 0, branchement en EPPL3)
EPPL3   BRX     EPPL2       (pour X = 1, branchement en EPPL1)
EPPL4   BRX     EPPL2       (pour X = 2, branchement en EPPL0)

        BRX     @EPDL15     (pour X = 0, branchement en EPPL1)
        BRX     @=EPDC15    (pour X = 2, branchement en EPPL2)
        BRX     @EPDL15     (pour X = 4, branchement en EPPL3)
        BRX     @=EPDC15    (pour X = 6, branchement en EPPL4)
```

BCT

NOM : Branchement si Carry vrai (Branch if Carry True)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C0	2,1	1,9
RM	C8	2,1	1,9
IL	D0	3,3	1,9
IG	D8	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si C = 1 ⟹ Y → (P)

Si C = 0 ⟹ (P) + 2 → (P)

Si l'indicateur Carry est égal à 1, l'adresse calculée Y est chargée dans le registre P ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si l'indicateur Carry est égal à 0, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
0		Poursuite en séquence
1		Branchement

Déroutements : standard

Exemples :

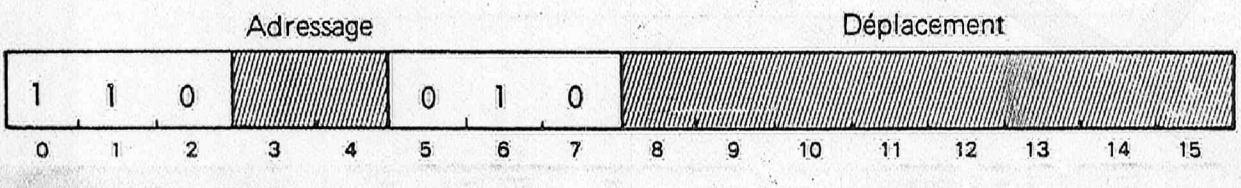
```
        BCT     @#EPDC14
EPPL2   BCT     $+1
EPPL3   BCT     EPPL3+1     → Equivalent à BCT $+1
        BCT     @EPDL13
```

BOT

NOM : Branchement si Overflow vrai (Branch if Overflow True)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C2	2,1	1,9
RM	CA	2,1	1,9
IL	D2	3,3	1,9
IG	DA	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si O = 1 ⟹ Y → (P)

Si O = 0 → (P) + 2 → (P)

Si l'indicateur Overflow est égal à 1, l'adresse calculée Y est chargée dans le registre P ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si l'indicateur Overflow est égal à 0, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
	0	Poursuivre en séquence
	1	Branchement

Déroutements : standard

Exemples :

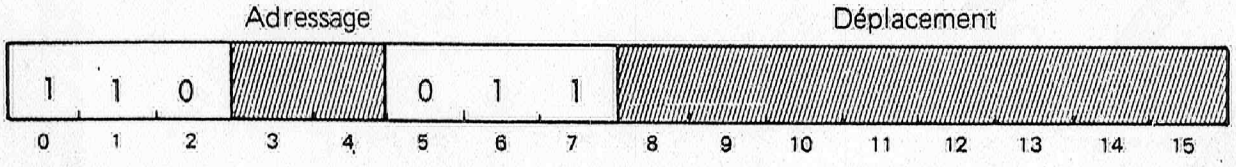
```
EPPL4    BOT    EPPL1
EPPL1    BOT    @ ≠EPDC14
EPPL3    BOT    @ EPDL13
EPPL2    BOT    EPPL4
```

BCF

NOM : Branchement si Carry faux (Branch if Carry False)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C3	2,1	1,9
RM	CB	2,1	1,9
IL	D3	3,3	1,9
IG	DB	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si C = 0 ⟶ Y ⟶ (P)

Si C = 1 ⟶ (P) + 2 ⟶ (P)

Si l'indicateur Carry est égal à 0, l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si l'indicateur Carry est égal à 1, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
0		Branchement
1		Poursuivre en séquence

Déroutements : standard

Exemples :

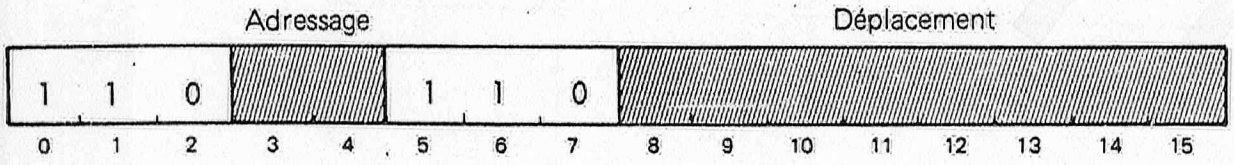
```
EPPL4    BCF    EPPL1
EPPL3    BCF    EPPL4
EPPL2    BCF    @EPDL14
EPPL1    BCF    @#EPDC13
```

BOF

NOM : Branchement si Overflow faux (Branch if Overflow False)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C6	2,1	1,9
RM	CE	2,1	1,9
IL	D6	3,3	1,9
IG	DE	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si O = 0 ⟶ Y ⟶ (P)

Si O = 1 ⟶ (P) + 2 ⟶ (P)

Si l'indicateur Overflow est égal à 0, l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si l'indicateur Overflow est égal à 1, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
	0	Branchement
	1	Poursuivre en séquence

Déroutements : standard

Exemples :

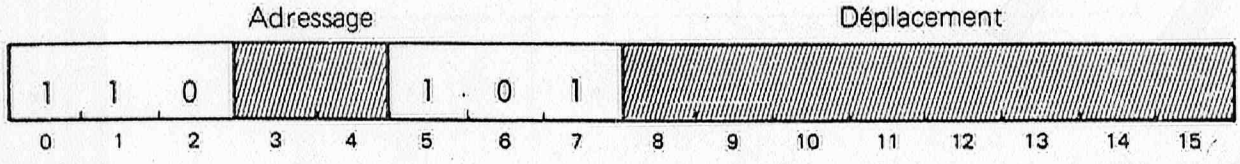
```
EPPL1    BOF    EPPL2+2
EPPL2    BOF    @EPDL14
EPPL3    BOF    @#EPDC13
         BOF    EPPL2-1 ⟶ Equivalent à BOF    EPPL1
```

BAZ

NOM : Branchement si A nul (Branch if A equal Zero)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C5	2,4	2,2
RM	CD	2,4	2,2
IL	D5	3,5	2,2
IG	DD	3,5	2,2

Si poursuite en séquence

Si branchement

Fonction : Si (A) = 0 ⟶ Y ⟶ (P)
Si (A) ≠ 0 ⟶ (P) + 2 ⟶ (P)

Si le contenu du registre A est égal à 0, l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si le contenu du registre A est différent de 0, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Déroutements : standard

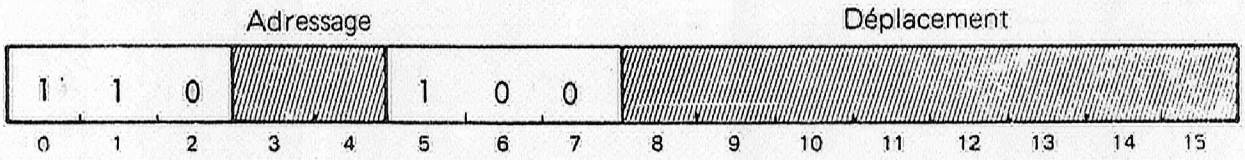
Exemples :

```
EPPL2   BAZ     @≠EPDC14
        BAZ     $+2
EPPL3   BAZ     $-1
        BAZ     @EPDL13
```

NOM : Branchement si A négatif (Branch on A Negative) **BAN**

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C4	2,4	2,2
RM	CC	2,4	2,2
IL	D4	3,5	2,2
IG	DC	3,5	2,2

Si poursuite en séquence

Si branchement

Fonction : Si (A) < 0 ⟶ Y ⟶ (P)
Si (A) ⩾ 0 ⟶ (P) + 2 ⟶ (P)

Si le contenu du registre A est inférieur à 0, l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si le contenu du registre A est égal ou supérieur à 0, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Déroutements : standard

Exemples :

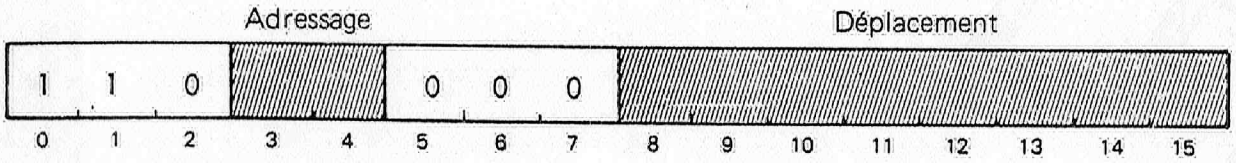
```
EPPL1   BAN     $+2
EPPL3   BAN     EPPL1
        BAN     @EPDL13
EPPL2   BAN     @#EPDC14
```

BE

NOM : Branchement si égal (Branch if Equal)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C0	2,1	1,9
RM	C8	2,1	1,9
IL	D0	3,3	1,9
IG	D8	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si Carry = 1 ⟹ Y ⟶ (P)

Si Carry = 0 ⟹ (P) + 2 ⟶ (P)

Cette instruction se situe normalement derrière une comparaison.

Si le premier terme de la comparaison était égal au second, l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si le premier terme de la comparaison était différent du second, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

O	C	Fonction suivant valeur initiale
0		Poursuivre en séquence
1		Branchement

Déroutements : standard

Divers : Cette instruction est équivalente à un BCT.

Exemples :

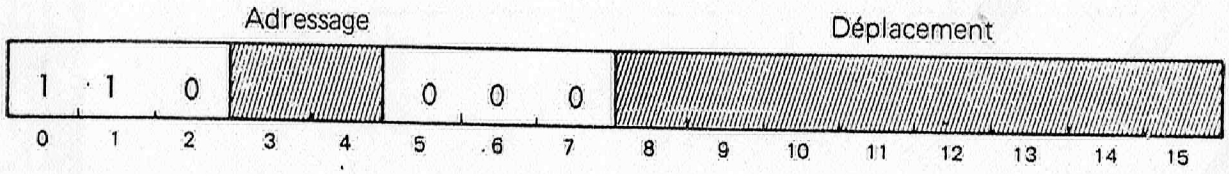
```
EPPL1    BE      EPPL4
EPPL4    BE      @#EPDC13
EPPL2    BE      @EPDL14
EPPL3    BE      EPPL1
```

BZ

NOM : Branchement si égal à zéro (Branch if equal Zero)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C0	2,1	1,9
RM	C8	2,1	1,9
IL	D0	3,3	1,9
IG	D8	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si Carry = 1 ⟹ Y ⟶ (P)

Si Carry = 0 ⟹ (P) + 2 ⟶ (P)

Cette instruction se situe normalement derrière une instruction de chargement.

Si la valeur précédemment chargée est égale à zéro, l'adresse calculée Y est chargée dans le registre P ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si la valeur précédemment chargée était différente de zéro, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
0		Poursuivre en séquence
1		Branchement

Déroutements : standard

Divers : Cette instruction est équivalente à un BCT

Exemples :

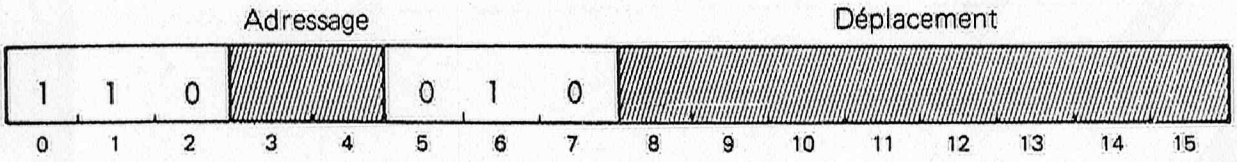
```
EPPL4    BZ      @EPDL14
EPPL2    BZ      EPPL1
EPPL3    BZ      @#EPDC13
EPPL1    BZ      EPPL4
```

BL

NOM : Branchement si inférieur (Branch if Less than)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C2	2,1	1,9
RM	CA	2,1	1,9
IL	D2	3,3	1,9
IG	DA	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si Overflow = 1 ⟹ Y → P

Si Overflow = 0 ⟹ (P) + 2 → (P)

Cette instruction se situe normalement derrière une comparaison.

Si le premier terme de la comparaison était inférieur au second, l'adresse calculée Y est chargée dans le registre P ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si le premier terme de la comparaison était égal ou supérieur au second, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
	0	Poursuivre en séquence
	1	Branchement

Déroutements : standard

Divers : Cette instruction est équivalente à un BOT

Exemples :

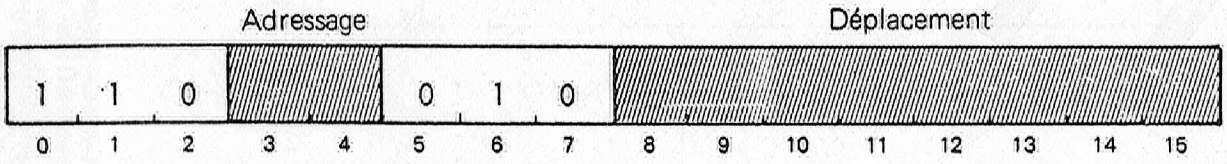
```
EPPL4    BL     @EPDL14
EPPL2    BL     EPPL1
EPPL3    BL     @#EPDC13
EPPL1    BL     EPPL4
```

BLZ

NOM : Branchement si négatif (Branch if Less than Zero)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C2	2,1	1,9
RM	CA	2,1	1,9
IL	D2	3,3	1,9
IG	DA	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si Overflow = 1 ⟹ Y ⟶ (P)

Si Overflow = 0 ⟹ (P) + 2 ⟶ (P)

Cette instruction se situe normalement derrière une instruction de chargement.

Si la valeur précédemment chargée était inférieure à zéro, l'adresse calculée Y est chargée dans le registre P ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si la valeur précédemment chargée était égale ou supérieure à zéro, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
	0	Poursuivre en séquence
	1	Branchement

Déroutements : standard

Divers : Cette instruction est équivalente à un BOT

Exemples :

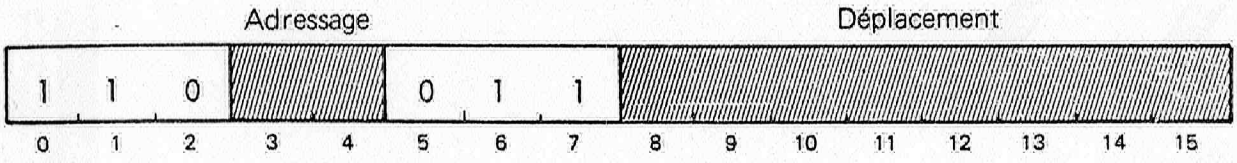
```
EPPL0    BLZ    @#EPDC14
EPPL2    BLZ    EPPL1
EPPL3    BLZ    @EPDL13
PL1      BLZ    $ - 3  ⟶ Equivalent à BLZ  EPPL0
```

BNE

NOM : Branchement si différent (Branch if Not Equal)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en us	
RP	C3	2,1	1,9
RM	CB	2,1	1,9
IL	D3	3,3	1,9
IG	DB	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si Carry = 0 ⇒ Y → (P)

Si Carry = 1 ⇒ (P) + 2 → (P)

Cette instruction se situe normalement derrière une comparaison.

Si le premier terme de la comparaison était différent du second, l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si le premier terme de la comparaison était égal au second, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
0		Branchement
1		Poursuivre en séquence

Déroutements : standard

Divers : Cette instruction est équivalente à un BCF

Exemples :

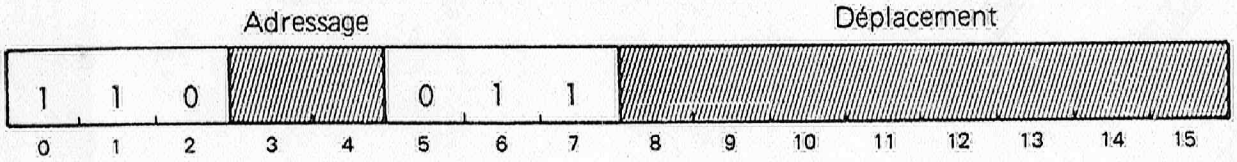
```
        BNE     $+1
        BNE     @EPDL13
EPPL2   BNE     @#EPDC14
EPPL3   BNE     $-3
```

BNZ

NOM : Branchement si non nul (Branch if Not equal Zero)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C3	2,1	1,9
RM	CB	2,1	1,9
IL	D3	3,3	1,9
IG	DB	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si Carry = 0 ⟹ Y ⟶ (P)

Si Carry = 1 ⟹ (P) + 2 ⟶ (P)

Cette instruction se situe normalement derrière une instruction de chargement.

Si la valeur précédemment chargée était différente de zéro, l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si la valeur précédemment chargée était égale à zéro, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

C	O	Fonction suivant valeur initiale
0		Branchement
1		Poursuivre en séquence

Déroutements : standard

Divers : Cette instruction est équivalente à un BCF

Exemples :

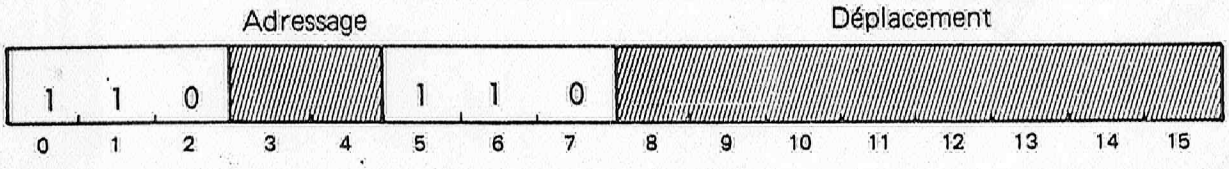
```
EPPL3    BNZ     EPPL1
EPPL4    BNZ     @EPDL13
EPPL2    BNZ     @#SPDC14
EPPL1    BNZ     EPPL4
```

BGE

NOM : Branchement si supérieur ou égal (Branch if Greater than or Equal to)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
RP	C6	2,1	1,9
RM	CE	2,1	1,9
IL	D6	3,3	1,9
IG	DE	3,3	1,9

Si poursuite en séquence

Si branchement

Fonction : Si Overflow = 0 ⟶ Y ⟶ (P)

Si Overflow = 1 ⟹ (P) + 2 ⟶ (P)

Cette instruction se situe normalement derrière une comparaison.

Si le premier terme de la comparaison était supérieur ou égal au second l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si le premier terme de la comparaison était inférieur au second, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

O	Fonction suivant valeur initiale
0	Branchement
1	Poursuivre en séquence

Déroutements : standard

Divers : Cette instruction est équivalente à un BOF

Exemples :

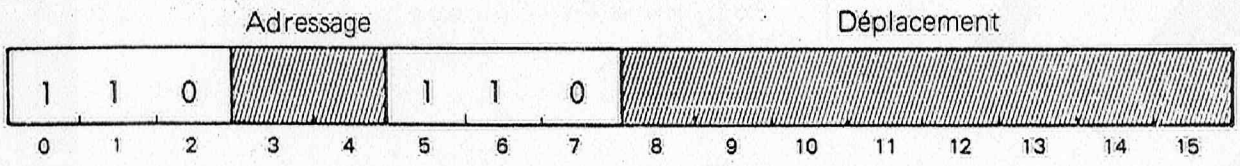
```
EPPL4    BGE    EPPL1
EPPL1    BGE    @EPDL13
EPPL2    BGE    @≠EPDC14
EPPL3    BGE    EPPL4
```

BPZ

NOM : Branchement si positif ou nul (Branch if Positive or equal Zero)

Classe : 2 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs	
		Si poursuite en séquence	Si branchement
RP	C6	2,1	1,9
RM	CE	2,1	1,9
IL	D6	3,3	1,9
IG	DE	3,3	1,9

Fonction : Si Overflow = 0 ⟹ Y ⟶ (P)

Si Overflow = 1 ⟹ (P) + 2 ⟶ (P)

Cette instruction se situe normalement derrière une instruction de chargement.

Si la valeur précédemment chargée était supérieure ou égale à zéro, l'adresse calculée Y est chargée dans le registre P, ce qui provoque la poursuite de l'exécution à partir de l'adresse Y.

Si la valeur précédemment chargée était inférieure à zéro, l'exécution se poursuit en séquence.

Eléments modifiés :

- Registres : P

Indicateurs :

O	Fonction suivant valeur initiale
0	Branchement
1	Poursuivre en séquence

Déroutements : standard

Divers : Cette instruction est équivalente à un BOF

Exemples :

```
EPPL2   BPZ     @EPDL14
EPPL3   BPZ     $+2
        BPZ     @#EPDC13
        BPZ     $-2  ⟶ Equivalent à BRU  EPPL3
```

VII-12. BRANCHEMENTS SYSTEME

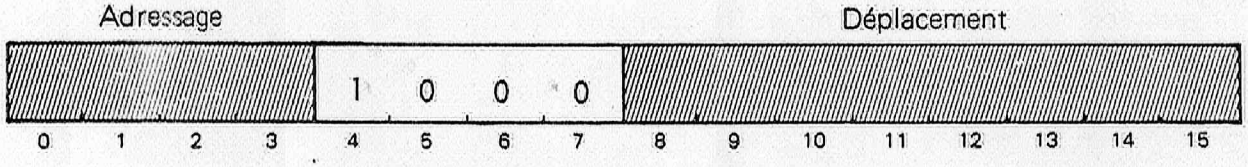
Il s'agit d'opérations évoluées d'appel et de communication entre les éléments du système : modules de programmes, modules moniteur, programmes d'interruption. Ce sont :

CLS	Call Section
RTS	Retour Section
CSV	Call Superviseur
RSV	Retour Superviseur
DIT	Désactivation d'interruption
DITR	Désactivation de l'interruption rapide

NOM : Appel section (CalL Section) **CLS**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en us	
DL	38	8,3	7,9
PX	E8	8,3	7,9
P	F8	8,3	7,9

→ En mode maître
→ En mode asservi

Fonction : (P) - G' → (((G) - 4 N) + (G))

(L) - G' → (((G) - 4 N) + (G) + 2)

(G) + ((G) - 4 N) → (L)

(G) + ((G) - 4 N + 2) → (P)

où N est l'opérande calculé c'est-à-dire (Y_2)

$N \neq 0$

Rappel

Un programme est constitué de sections, chaque section étant caractérisée par une valeur de la base L et une valeur de la base P. Ces valeurs caractéristiques sont stockées dans la PRT du programme (un SRD par section).

But du Call section

Le but du CLS est d'effectuer un branchement entre une section (appelante) et une autre du même programme (appelée) tout en assurant des possibilités de communication permanentes entre la section appelée et la section appelante ainsi qu'un retour simple.

Eléments utilisés

- PRT du programme
- Deux premiers mots du LDS de la section appelée
- Contenu de l'adresse calculée de l'instruction CLS, cette adresse contenant le numéro de la section appelée. (Signalons que les assembleurs et compilateurs associés à un éditeur de liens offrent la possibilité d'appeler les sections par un nom, ce qui permet à l'utilisateur d'ignorer le numéro réel de la section qu'il appelle).

Fonctionnement du Call section

- Rangement dans les deux premiers mots du LDS de la section appelée des valeurs relatives à G des bases L et P de la section appelante.
- Recherche à l'aide du numéro de section des valeurs L et P de la section appelée et rangement dans les registres correspondants.
- Branchement sur la section appelée.

Communication avec la section appelante

Il existe trois méthodes pour transférer les paramètres entre la section appelante et la section appelée :

1) A travers la section de données communes (CDS)

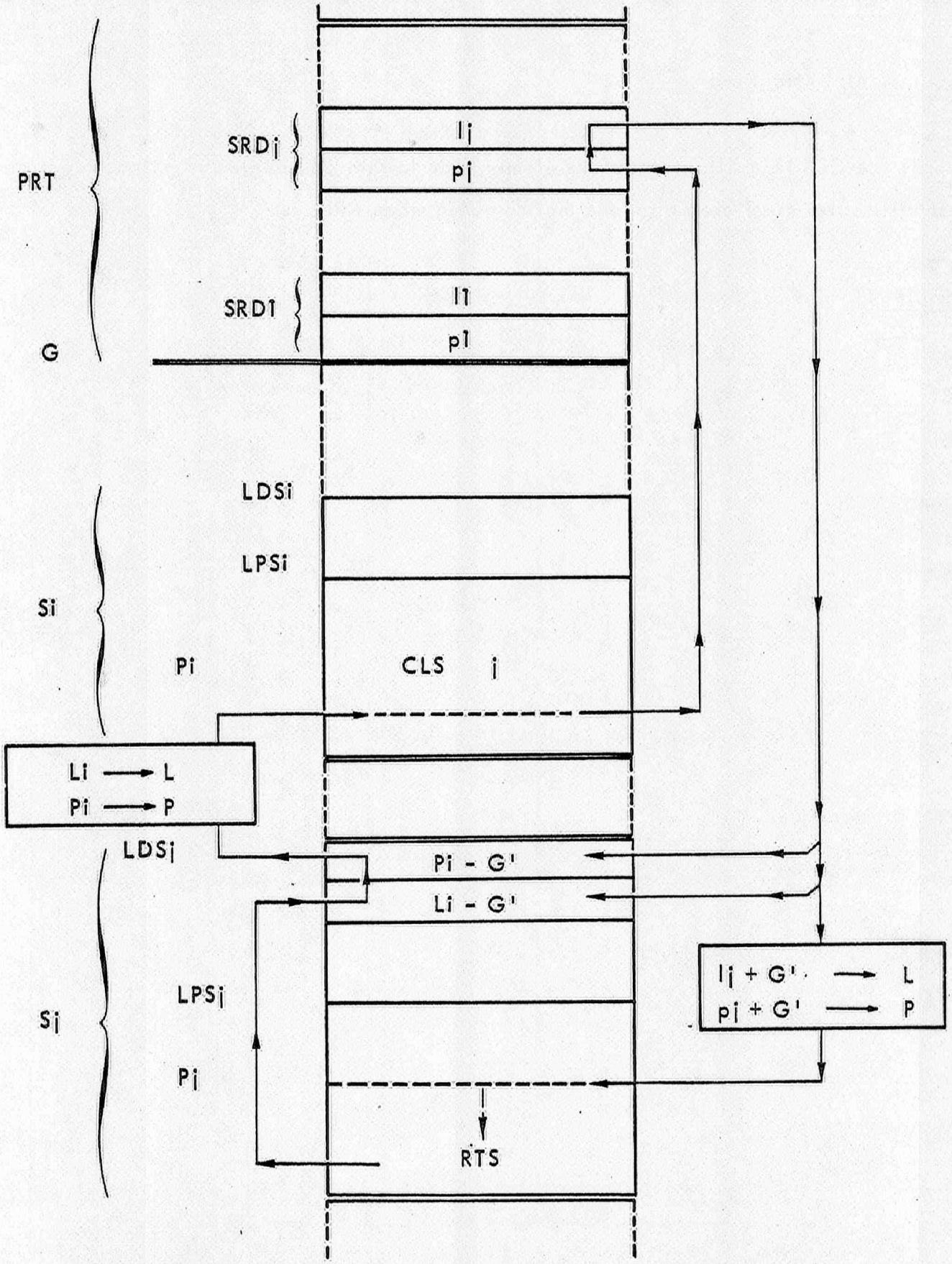
2) En mode indirect local ou indirect local indexé par le deuxième mot du LDS.

3) Par l'intermédiaire des registres A, E, X et/ou des indicateurs C et O.

Schéma de fonctionnement

j = numéro de la section appelée

i = numéro de la section appelante



Eléments modifiés :

- Registres : L - P

- Positions mémoire : Les deux premiers mots du LDS de la section appelée

Déroutements : standard

Divers : Le CLS travaillant avec des éléments de la section appelée n'est pas réentrant. Il est utilisables aussi bien en mode maître qu'en mode esclave.

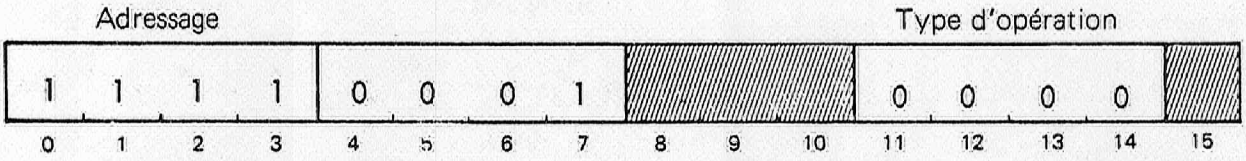
Exemples :

```
CLS         LPS1
CLS         =2
CLS         =2,x
```

NOM : Retour section (ReTurn Section) **RTS**

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution	
P	F1	00	4,3	4,7

En mode maître
En mode asservi

Fonction : ((L)) + G' + 2 → (P)
((L) + 2) + G' → (L)

Le RTS situé dans une section appelée par un CLS assure le retour à la section appelante en restaurant dans les registres L et P des valeurs correspondantes se trouvant dans les deux premiers mots du LDS de la section appelée.

Le schéma de fonctionnement du RTS est indiqué sur le schéma de fonctionnement du CLS.

Eléments modifiés :

- Registres : L - P

Déroutements : standard

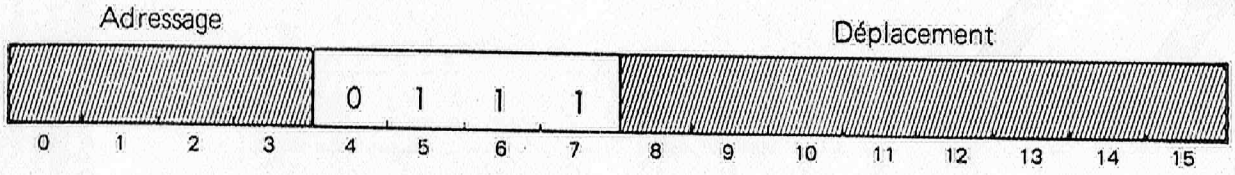
Exemples : RTS

CSV

NOM : Appel superviseur (Call SuperVisor)

Classe : 1 Privilégiée : non Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	37	9,9
PX	E7	9,9
P	F7	9,9

Fonction : (P) - (G) → ((G))

(L) - (G) → ((G) + 2)

Indicateurs → ((G) + 4)

1 → PR

1 → MS

((12) - 4 N) → (L)

((12) - 4 N + 2) → (P)

ou N est l'opérande calculé c'est-à-dire (Y_2)

Rappel

Un moniteur est composé de sections, chaque section étant caractérisé par une valeur de la base L et une valeur de la base P. Ces valeurs caractéristiques sont stockées dans la PRTS (PRT du superviseur) située à un emplacement quelconque de la mémoire, et pointée par l'intermédiaire du mot d'adresse absolue 12 (décimal). Un moniteur fonctionne en mode maître et protégé.

But du Call Superviseur

Le but du CSV est d'effectuer un branchement entre une section d'un programme utilisateur et une section superviseur tout en assurant la réentrance de la section superviseur, ainsi qu'un retour simple au programme utilisateur.

Eléments utilisés

- PRTS (PRT du superviseur)
- Pointeur de la PRTS
- Trois premiers mots de la CDS du programme appelant
- Contenu de l'adresse calculée de l'instruction CSV, cette adresse contenant le numéro de la section appelée. (Signalons que les assembleurs et compilateurs associés à un éditeur de liens offrent la possibilité d'appeler des sections superviseur par un nom de la forme M:xxxx, ce qui permet à l'utilisateur d'ignorer le numéro réel de la section qu'il appelle).

Fonctionnement du Call Superviseur

- Rangement dans les trois premiers mots de la CDS du programme appelant des valeurs relatives à G des bases L et P de la section en cours ainsi que des valeurs des indicateurs.
- Recherche à l'aide du numéro de section des valeurs L et P de la section appelée
- Forçage à 1 des indicateurs MA et PR
- Branchement sur la section appelée

Communication avec la section appelante

Les communications se font en mode indirect général indexé par le deuxième mot de la CDS du programme appelant.

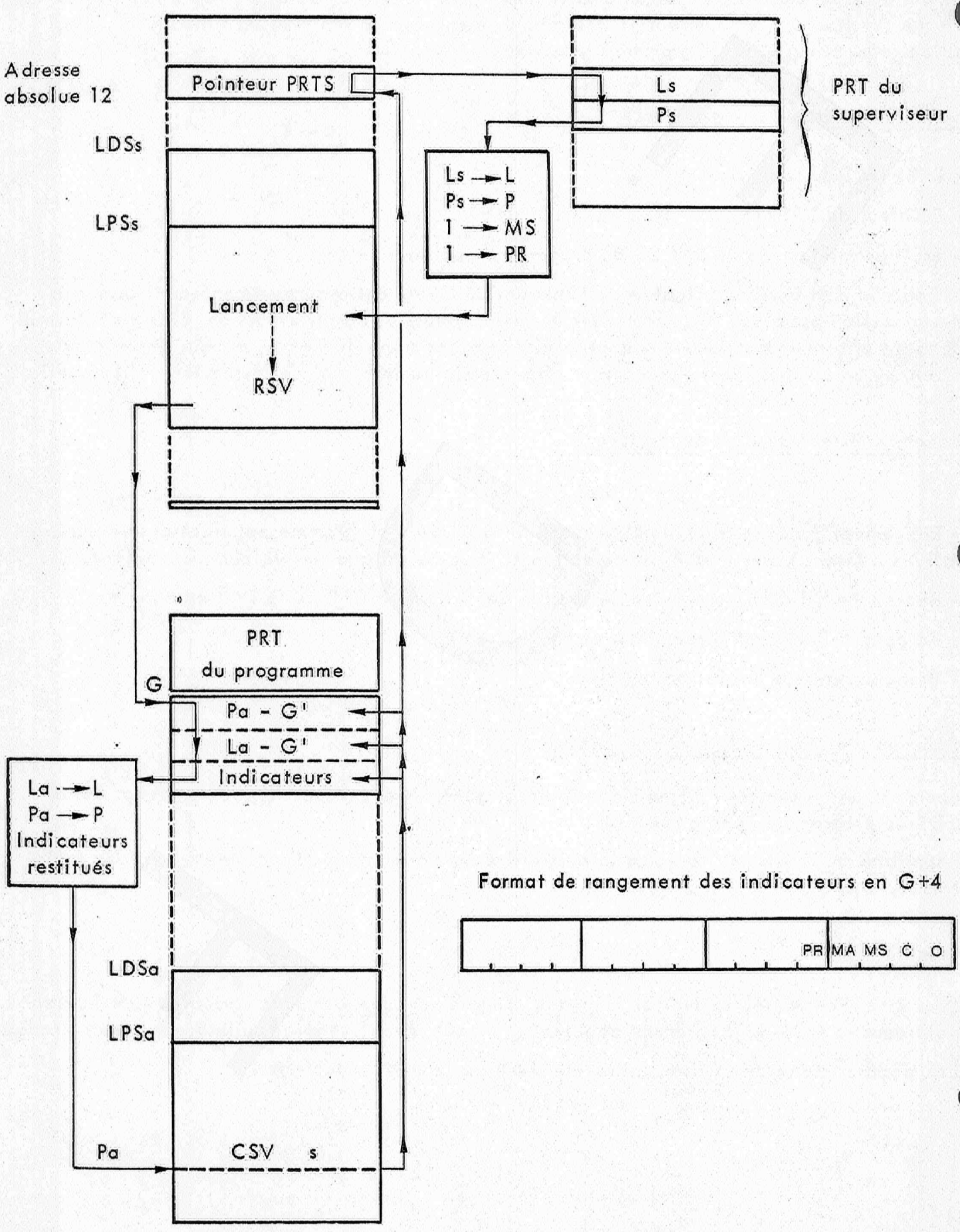
Rappelons qu'il subsiste toujours la possibilité de transmettre des paramètres par les registres A, E et X.

Réentrance

Pour être réentrante, la section moniteur rangera tous les éléments variables qu'elle manipule dans la CDS du programme appelant (en mode direct général ou indirect général indexé).

Les modules standard utilisent ainsi une TWB de 16 mots à cet effet.

Schéma de fonctionnement :



Format de rangement des indicateurs en G+4

Eléments modifiés :

- Registres : L - P
- Positions mémoire : Les trois premiers mots de la CDS du programme appelant. Soit (G) à (G + 5)
- Indicateurs : MS est mis à 1
  PR est mis à 1

Déroutements : standard

Divers : Le programme n'est pas interruptible entre un CSV et l'instruction exécutée immédiatement après (afin de permettre un masquage immédiat des interruptions si nécessaire).

Exemples :

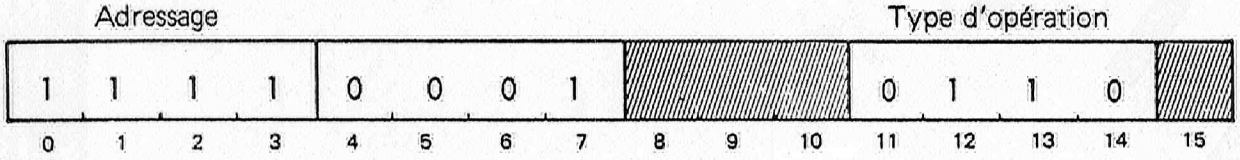
```
CSV     M:IO
CSV     =2
CSV     =2,x
```

RSV

NOM : Retour superviseur (Return SuperVisor)

Classe : 1 Privilégiée : oui Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
P	F1C0	6,7

Fonction : ((G) + 4) → Indicateurs

(G) + ((G) + 2) → (L)

(G) + 2 + ((G)) → (P)

Le RSV, situé dans une section moniteur (mode maître) appelée par un CSV assure le retour à la section appelante en restaurant dans les registres L et P les valeurs correspondantes se trouvant dans les deux premiers mots de la CDS du programme auquel appartient la section appelante. Il restaure également les indicateurs dans leur état initial.

Le schéma de fonctionnement du RSV est indiqué sur le schéma de fonctionnement du CSV.

Eléments modifiés :

- Registres : L - P
- Indicateurs : Tous les indicateurs sont restitués (valeur qu'ils avaient avant le CSV si la base G n'a pas été modifiée entre temps).

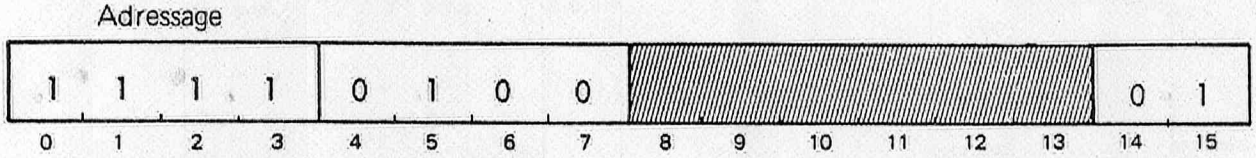
Déroutements : Violation de mode et standard

Exemples : RSV

NOM : Désactiver IT (Desactivate IT) **DIT**

Classe : 1 Privilégiée : oui Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
P	F4	32,5

Fonction :

Rappel

On appelle niveau d'un programme, le niveau de l'interruption qui le déclenche. Un programme déclenché par aucune interruption est dit au niveau zéro.

Le système d'interruption MITRA 15 est organisé en niveaux hiérarchisés.

Toute arrivée d'une interruption d'un niveau supérieur à celui du programme en cours, interromps celui-ci. Une interruption d'un niveau inférieur ou égal au niveau du programme en cours est alors mise en attente jusqu'à désactivation du niveau plus prioritaire.

But de l'instruction DIT

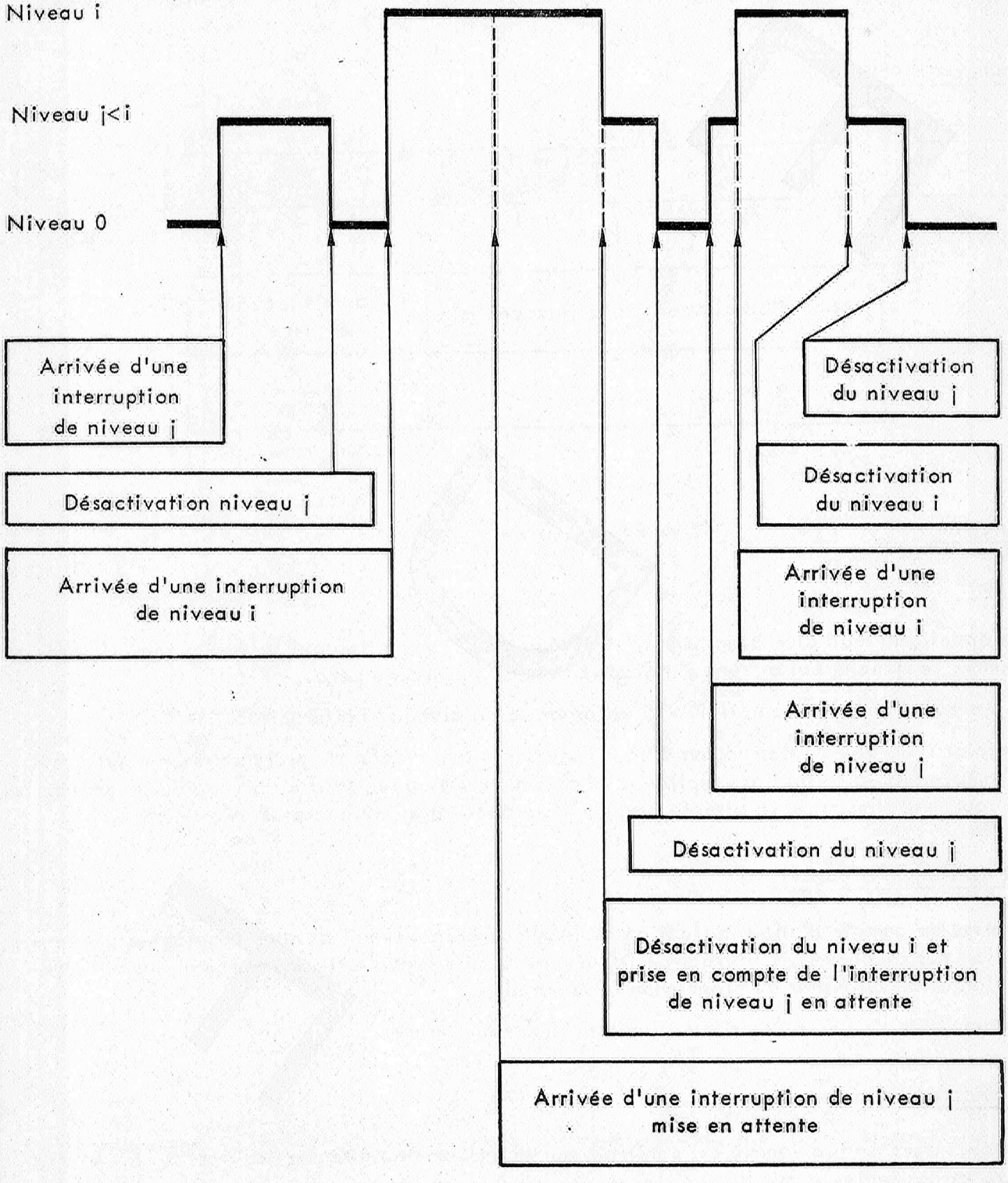
La prise en compte d'une interruption réalisant un branchement au sous-programme d'interruption correspondant, l'instruction DIT a pour but de terminer le sous-programme d'interruption et de retourner au programme interrompu.

Il s'agit bien de ce point de vue d'un branchement système.

Eléments utilisés :

Le contexte d'un programme est constitué par la valeur courante des registres X, E, A, G, L, P, et des indicateurs.

Exemple de déroulement



Pour chaque niveau d'interruption, il existe un pointeur de la zone mémoire où pourra éventuellement être préservé le contexte en cas d'interruption. La zone mémoire de préservation du contexte d'un niveau n'a d'existence réelle que s'il existe un programme connecté à ce niveau. Cette zone est située généralement immédiatement derrière le programme proprement dit.

Cette zone contient les valeurs des registres et indicateurs au moment où le programme a été la dernière fois interrompu (soit par prise en compte d'une interruption plus prioritaire, soit par désactivation de son niveau). Si le programme n'a jamais été interrompu, la zone de préservation contient le contexte initial du programme (valeurs au moment du lancement).

Les 32 pointeurs de contexte (pointeurs de la zone de préservation associée à chaque niveau) sont rangés par valeurs de niveau croissantes dans la table des pointeurs de contexte (par adresses croissantes à partir de l'adresse CPT).

Les éléments préservés sont rangés à raison d'un élément par mot, par adresses croissantes à partir de l'adresse pointée par le pointeur de contexte dans l'ordre suivant :

- Indicateurs, X, E, A, G, L, P

Fonctionnement du DIT

- Incrémentation de 2 de la valeur de P
- Désactivation du niveau en cours
- Rangement du contexte à partir de l'adresse mémoire pointée par le pointeur de contexte du niveau correspondant.
- Prise en compte du niveau le plus prioritaire en attente (mise à jour de R8)
- Chargement du contexte avec le contenu de la zone de préservation pointée par le pointeur de contexte du nouveau niveau (ce qui équivaut à un lancement).

Prise en compte d'une interruption

Le DIT effectue le branchement de retour du sous-programme d'interruption, l'appel de ce sous-programme était réalisé par la prise en compte de l'interruption correspondante qui effectue :

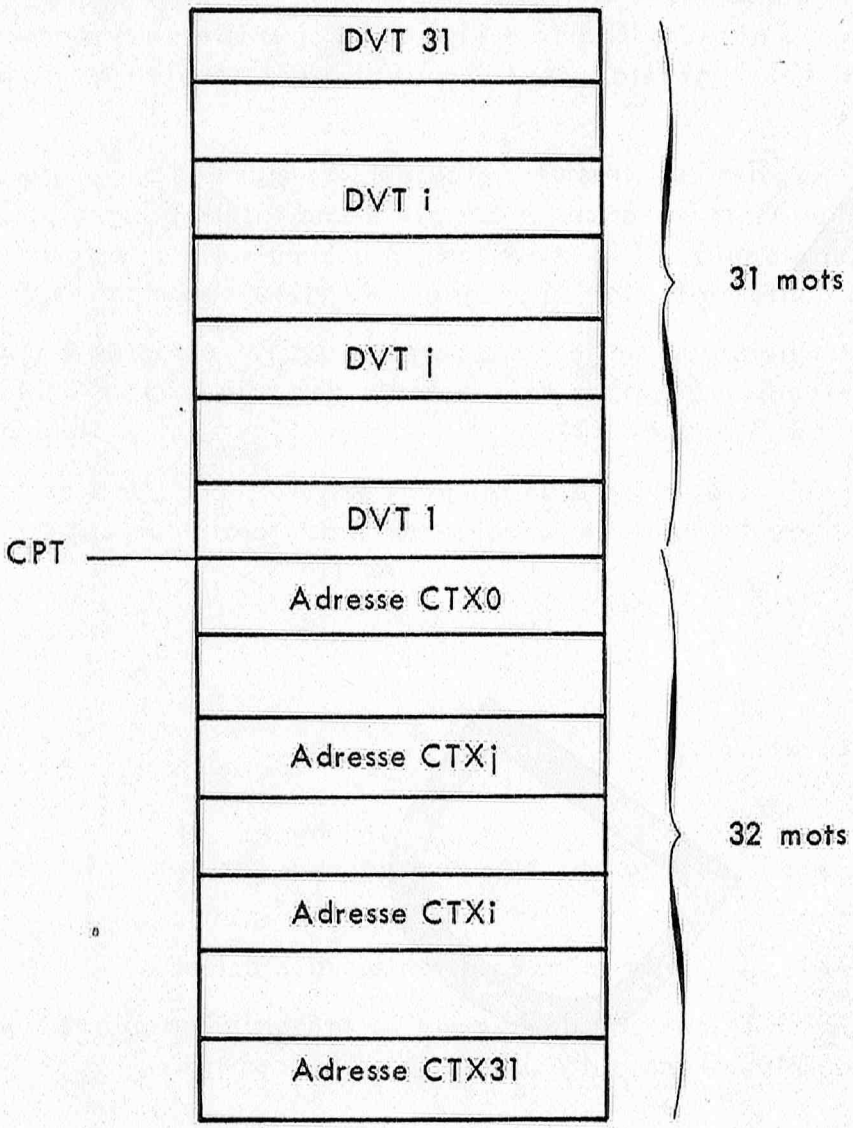
- Rangement du contexte du niveau interrompu à partir de l'adresse mémoire pointée par le pointeur de contexte du niveau considéré.
- Prise en compte du niveau de l'interruption prioritaire (mise à jour de R8)
- Chargement du contexte avec le contenu de la zone de préservation pointée par le pointeur de contexte du nouveau niveau (ce qui équivaut à un lancement).

Configurations d'interruption

Il y a 31 configurations d'interruption (il n'y en a pas évidemment pour le niveau zéro). Ces configurations sont utilisées par le DIT pour connaître l'implantation exacte de l'interruption à désactiver.

Elles sont rangées par valeurs de niveau croissantes dans la table DVT (par adresses décroissan-

tes à partir de l'adresse CPT).



où i > j

Cette double table peut être un emplacement quelconque de la mémoire, l'adresse CPT se trouvant dans le mot d'adresse absolue 10 (décimal).

Niveau du programme en cours

Le niveau du programme en cours est repéré par le registre R8 qui contient le double de ce niveau.

Eléments modifiés :

- Registres : A, E, X, P, L, G, R8
- Positions mémoire : Les 8 mots du contexte de l'interruption désactivée
- Indicateurs : Tous les indicateurs

Déroutements : Violation de mode et standard

Divers : Le DIT assurant la permutation des contextes, aucune protection ni aucun masquage ne sont nécessaires.

Tout programme d'interruption doit se terminer par un DIT.

Un DIT n'a pas de sens au niveau zéro (on ne peut pas désactiver le niveau zéro)

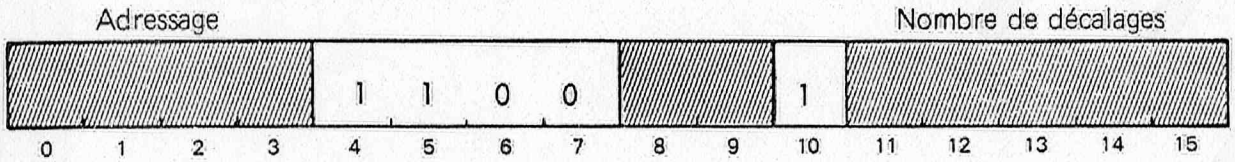
Exemples : DIT

DITR

NOM : Désactivation d'IT rapide

Classe : 1 Privilégiée : oui Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal		Temps d'exécution en µs
PX	EC	20	6,1
P	FC	20	6,1

Fonction :

Comparaison entre interruption normale et interruption rapide

- La prise en compte d'une interruption normale comprend les opérations suivantes :

- Sauvegarde en mémoire du contexte en cours dont le niveau est repéré par R8
- Prise en compte du niveau appelant (Na) et mise à jour de R8
- Chargement du contexte avec les éléments correspondants à Na

- L'acquittement d'une interruption normale comprend les opérations suivantes :

Le sous-programme d'interruption se termine par l'instruction DIT qui réalise :

- Sauvegarde en mémoire du contexte correspondant à l'interruption traitée
- Désactivation de l'interruption traitée
- Prise en compte du niveau en attente et mise à jour de R8
- Chargement du contexte avec les éléments correspondants au nouveau niveau

- La prise en compte de l'interruption rapide comprend les opérations suivantes :

- Mise en attente des interruptions normales tant que l'interruption rapide n'est pas acquittée.

- Sauvegarde des indicateurs en cours dans le registre 6 du bloc 0
- Chargement dans R12 du numéro de bloc réservé à l'interruption rapide
- Chargement des indicateurs à partir du registre 6 du bloc réservé à l'interruption rapide

. L'acquittement de l'IT rapide comprend les opérations suivantes :

Le sous-programme d'interruption rapide se termine par l'instruction DITR qui réalise

- Sauvegarde des indicateurs dans le registre 6 du bloc réservé à l'interruption rapide
- Mise à zéro de R12 (Retour au bloc zéro)
- Désactivation de l'interruption rapide (d'où libération des interruptions normales)
- Restitution des indicateurs précédemment rangés dans le registre 6 du bloc 0.

Eléments modifiés :

- Registres : Retour au bloc 0
- Indicateurs : Restitution

Déroutements : Violation de mode et standard

Exemples : DITR

VII-13. INSTRUCTIONS DE COMMANDE

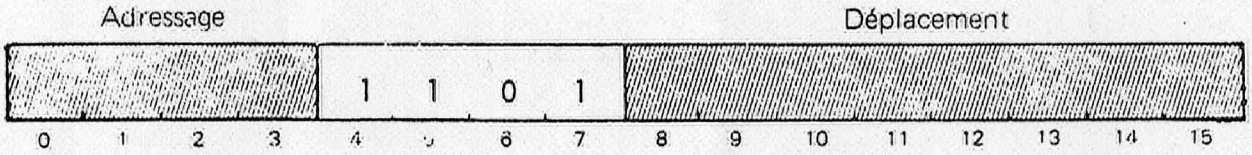
Ce sont :

TES	(Test ans Set)	Test et marquage
STM	(Set mask)	Masquage des interruptions
CLM	(Clear mask)	Démasquage des interruptions
RD	(Read Direct)	Lecture directe
WD	(Write Direct)	Ecriture directe
LDP	(Load Protection)	Chargement de la protection mémoire

NOM : Marquage et Test (TEst and Set) **TES**

Classe : 1 Privilégiée : oui Optionnelle : oui

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	3D	3,6
PX	ED	3,4
P	FD	3,4

Fonction :

P : $\begin{cases} (De)_2 \longrightarrow A \\ 0 \longrightarrow (De)_2 \end{cases}$

PX : $\begin{cases} (De + (X))_2 \longrightarrow A \\ 0 \longrightarrow (De + (X))_2 \end{cases}$

DL : $\begin{cases} ((De + (L)))_2 \longrightarrow A \\ 0 \longrightarrow ((De + (L)))_2 \end{cases}$

Cette instruction permet de tester une position mémoire et de la forcer à zéro sans être interrompu. La valeur initiale est rangée dans A. Le résultat du test est chargé dans les indicateurs. Cette instruction est nécessaire dans le cas où plusieurs Unités de Traitement utilisent une mémoire commune. Elle permet par exemple d'assurer le contrôle d'occupation de tables par l'un ou l'autre processeur.

Le bit protection est également mis à zéro.

Eléments modifiés :

- Registres : A
- Positions mémoire : y_2
- Indicateurs : C - O

Indicateurs :

C	O	Après exécution
0	0	$y_2 < 0$
0	1	$y_2 > 0$
1	0	$y_2 = 0$

Déroutements : Violation de mode
Instruction inexistante et standard

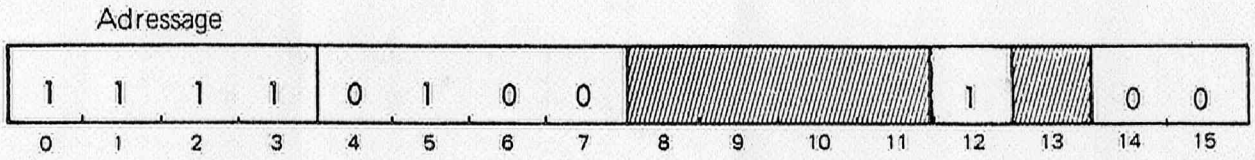
Exemples :

```
TES     EPDL28
TES     =& 3E,x
TES     =9
```

NOM : Masquage des interruptions (SeT Mask) **STM**

Classe : 1 Privilégiée : oui Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
P	F4	3,4

Fonction : $(N_{12}) \longrightarrow$ Masque des interruptions (MA)

La valeur 1 est rangée dans l'indicateur MA.

Ceci a pour effet de masquer toutes les interruptions.

Eléments modifiés :

- Indicateurs : MA

Déroutements : Violation de mode et standard

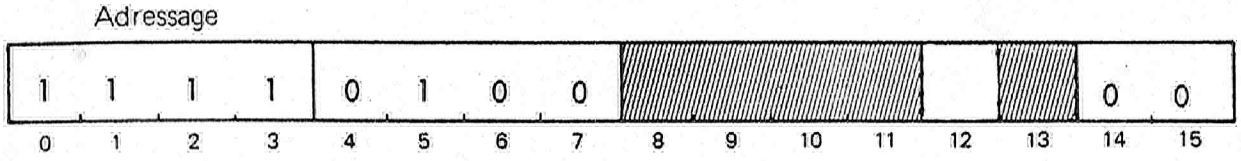
Exemples : STM

CLM

NOM : Démasquage des interruptions (CLear Mask)

Classe : 1 Privilégiée : oui Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
P	F4	3,4

Fonction : $(N_{12}) \rightarrow$ Masque des interruptions (MA)

La valeur 0 est rangé dans l'indicateur MA. Ceci a pour effet de démasquer toutes les interruptions.

Eléments modifiés :

- Indicateurs : MA

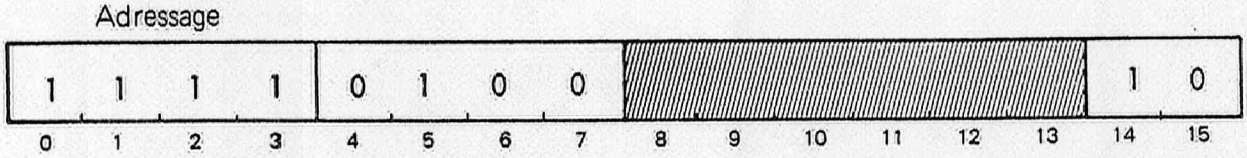
Déroutements : Violation de mode et standard

Exemples : CLM

NOM : Lecture directe (Read Direct) **RD**

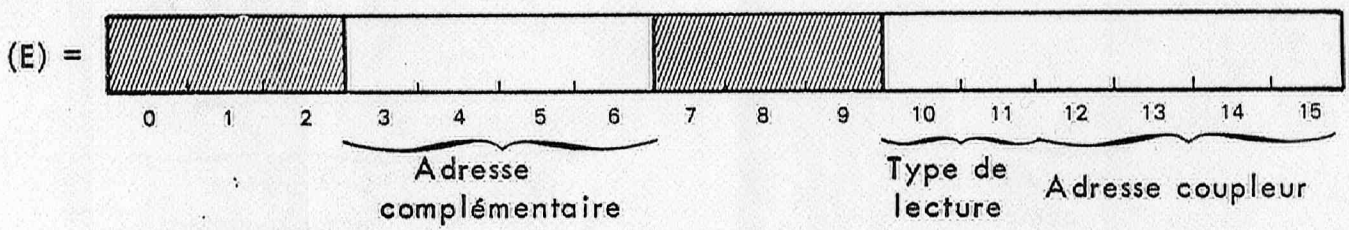
Classe : 1 Privilégiée : oui Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
P	F4	3,5

Fonction : L'action est déterminée par (E)



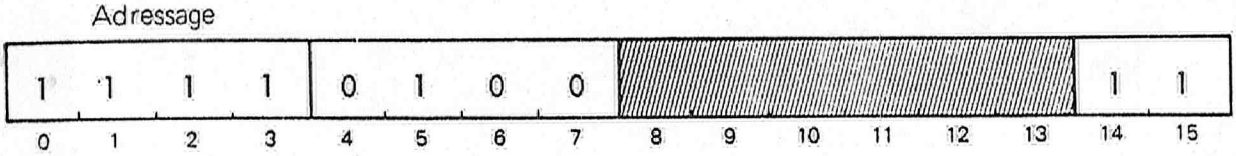
C'est l'instruction d'entrée dont le sens dépend du coupleur adressé

WD

NOM : Ecriture directe (Write Direct)

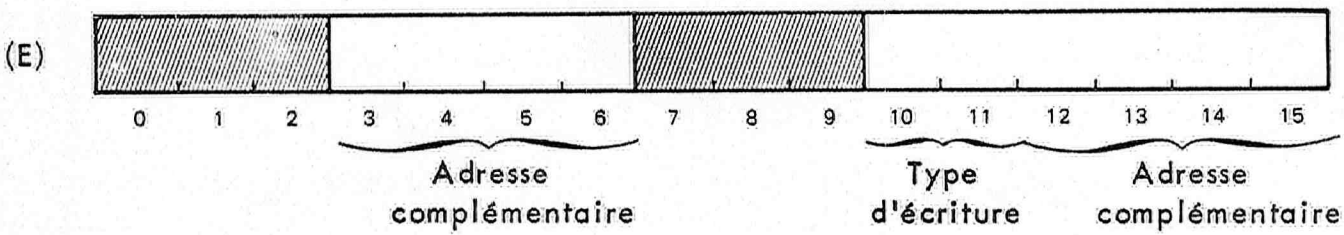
Classe : 1 Privilégiée : oui Optionnelle : non

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
P	F4	3,5

Fonction : L'action est déterminée par (E)

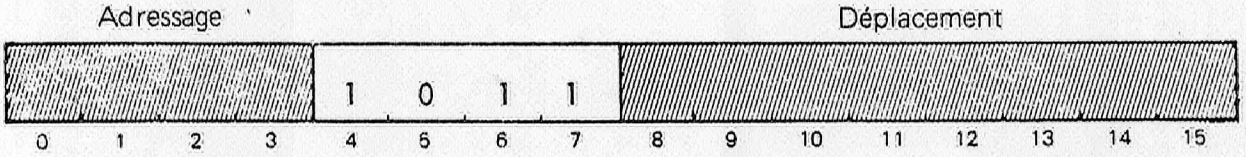


C'est l'instruction de sortie dont le sens dépend du coupleur adressé.

NOM : Chargement protection mémoire (LoaD Protection) **LDP**

Classe : 1 Privilégiée : oui Optionnelle : oui
Mode passe-partout uniquement (PM=1)

Code de l'instruction :



Mode d'adressage	Code Hexadécimal	Temps d'exécution en µs
DL	3B	3,1 + 1,8 n
PX	EB	3,1 + 1,4 n
P	FB	3,1 + 1,4 n

n = nombre de mots de la chaine

Fonction : Pour X variant de 0 à (E) - 1

$N_{15} \longrightarrow bp\ ((A) + 2X)$

En fin d'exécution E = -1
A = Adresse du premier mot non traité

L'instruction LDP charge le bit de protection d'une chaîne de mots.

L'adresse du premier mot de cette chaîne se trouve dans A. La longueur de cette chaîne exprimée en nombre de mots se trouve dans E.

La valeur chargée dans chaque bit de protection est la valeur du bit 15 de l'opérande calculé (contenu de l'adresse calculée). La chaîne devient protégée si ce bit vaut 1. Elle devient non protégée dans le cas contraire.

Eléments modifiés :

- Registres : E - A

- Positions mémoire : (y_2) à $(y_2 + (E) - 1)$

Déroutements : Violation de mode et standard

Divers : Cette instruction est interruptible entre chaque mot

Exemples :

LDP	=1	Protection
LDP	=0	Libération
LDP	=0,x	
LDP	EPDL21	

annexe A_Liste des directives

Les directives d'assemblage sont représentées dans le tableau ci-dessous avec les conventions suivantes :

- Directive : Nom de la directive
- Format : Format de l'opérande
- Type : Type de l'opérande :

. EE expression d'étiquette
. ER expression de référence
. V valeur
. C chaîne de caractères
. S nom de segment

- Fonction : Définition de la fonction de cette directive
- DON : Directive autorisée dans un CDS ou un LDS
- PROG : Directive autorisée dans un LPS
- ETIQ : Cette directive donne une valeur d'adresse à une étiquette

DIRECTIVE	FORMAT	Type	FONCTION	DON	PROG	ETIQ
CDS	[DUM]		La directive CDS définie la section de données communes CDS. - DUM aucun code n'est généré.			x
LDS	[DUM]		La directive LDS définie un segment de données locales LDS. - DUM aucun code n'est généré.			x
IDS (MITRAS II seulement)	[DUM]		La directive IDS identifie un segment de données à accès indirect. Toute étiquette apparaissant dans ce segment est définie en valeur relative au début du segment. - DUM aucun code n'est généré.	x	x	x
LPS	NOM	S	La directive LPS définie un segment de programme exécutable LPS et, par là, une section de programme. - NOM nom de la LDS associée à cette LPS.			x
END	NOM DE SECTION	S	La directive END indique la fin du module d'assemblage. - NOM DE SECTION nom de la section de lancement.			
FIN	[ETIQUETTE]	EE	La directive FIN indique la fin d'un segment. - ETIQUETTE Ne peut être présente que dans le cas d'un LPS. Dans ce cas elle définie l'adresse de lancement dans ce segment.	x	x	
RES ,1	VALEUR	V	La directive RES permet la réservation d'une zone mémoire. - ,1 Si cette option est présente dans la zone commande, l'unité d'emplacement est l'octet; sinon c'est le mot. - VALEUR Indique la taille de la zone mémoire réservée.	x	x	x
BND			La directive BND permet l'avancement du compteur d'emplacement à une limite de mot.	x	x	

DIRECTIVE	FORMAT	Type	FONCTION	DON	PROG	ETIQ
EQU	{Expression prédéfinie / $}	EE	La directive EQU permet l'équivalence entre le symbole se trouvant en zone étiquette et la quantité définie en zone opérande.	x	x	x
GOTO,n (MITRAS II seulement)	Etiq1,Etiq2,... Etiqn	EE	La directive GOTO permet d'effectuer un branchement conditionnel à l'assemblage. - ,n Valeur pointant l'étiquette située en zone opérande sur laquelle doit s'effectuer le branchement.	x	x	
DO (MITRAS II seulement)	VALEUR		La directive DO permet d'effectuer une itération d'une instruction à l'assemblage. - VALEUR Indique le nombre d'itérations - % à chaque itération, ce caractère prend la valeur du compteur d'itération.	x		
DATA ,1	[≠] Expression 1 [, [≠] Expression 2]...	EE V	La directive DATA permet la génération de données - ,1 Si cette option est présente dans la zone commande, l'unité d'emplacement est l'octet, sinon c'est le mot. - = Ce symbole indique que l'expression qui suit doit rester relative à la base G lors d'un chargement en mode maître.	x		x
GEN, Liste de zones (MITRAS II seulement)	Liste d'expressions	V	La directive GEN permet la génération de valeur. - Liste de zone C'est une liste de valeur dont chacune d'elle définie la taille d'une zone à générer. La somme de ces valeurs doit être égale à 8 ou 16. - Liste d'expressions C'est une liste d'expressions qui définissent le contenu de chacune des zones déclarées.	x	x	x
TEXT	"chaîne de caractères"	C	La directive TEXT permet la génération d'une chaîne de caractères. - Chaine de caractères Est constituée de caractères alphanumériques.	x		x
DEF	Etiquette[,Etiquette]..	EE	La directive DEF permet de déclarer des étiquettes comme définitions externes.	x	x	
REF	[≠] Etiquette [, [≠] Etiquette] ...	EE	La directive REF permet de déclarer des étiquettes comme références externes. - = Indique que l'étiquette fait partie de la CDS.	x	x	
BASE	[Etiquette]		La directive BASE permet, lors de l'assemblage, de générer toutes les adresses en valeur relative à l'adresse définie en zone opérande.		x	
PAGE (MITRA II seulement)			La directive PAGE permet d'imprimer la liste d'assemblage sur la page suivante si l'organe de sortie est une imprimante.	x	x	

annexe B_Liste des instructions

Liste des instructions par ordre alphabétique

Instr.	Classe	Code suivant adressage							
		P	DL	IL	ILX	DG	IGX	RP	RM
ADD	0	25	05	65	A5	45	85	-	-
ADM	0'	-	17	77	B7	57	97	-	-
AND	0	29	09	69	A9	49	89	-	-
BAN	2	-	-	D4	-	-	DC	C4	CC
BAZ	2	-	-	D5	-	-	DD	C5	CD
BCF	2	-	-	D3	-	-	D8	C3	C8
BCT	2	-	-	D0	-	-	D8	C0	C8
BOF	2	-	-	D6	-	-	DE	C6	CE
BOT	2	-	-	D2	-	-	DA	C2	CA
BRU	2	-	-	D7	-	-	DF	C7	CF
BRX	2	-	-	D1	-	-	D9	C1	C9
• CLM	1	F400	-	-	-	-	-	-	-
CLS	1	F8	38	-	-	-	-	E8°o	-
CMP	0	2B	0B	6B	AB	4B	8B	-	-
*CPS	0	2A	0A	6A	AA	4A	8A	-	-
CSV	1	F7	37	-	-	-	-	E7 o	-
DCL	1	F6	36	-	-	-	-	E6 o	-
DCX	1	F3	33	-	-	-	-	E3 o	-
• DIT	1	F401	-	-	-	-	-	-	-
*DIV	0	28	08	68	A8	48	88	-	-
DLD	0'	-	10	70	80	50	90	-	-
DST	0'	-	16	76	86	56	96	-	-
EOR	0	23	03	63	A3	43	83	-	-
*FAD	0'	-	1A	7A	BA	5A	9A	-	-
*FDV	0'	-	1D	7D	BD	5D	9D	-	-
*FMU	0'	-	1C	7C	BC	5C	9C	-	-
*FSU	0'	-	1B	7B	BB	5B	9B	-	-
ICL	1	F5	35	-	-	-	-	E5 o	-
ICX	1	F2	32	-	-	-	-	E2 o	-
IOR	0	27	07	67	A7	47	87	-	-

Instr.	Classe	Code suivant adressage							
		P	DL	IL	ILX	DG	IGX	RP	RM
LBL	0	2D	0D	6D	AD	4D	8D	-	-
LBR	0	2E	0E	6E	AE	4E	8E	-	-
LBX	0	2F	0F	6F	AF	4F	8F	-	-
LDA	0	20	00	60	A0	40	80	-	-
LDE	0	21	01	61	A1	41	81	-	-
*•LDP	1	FB	3B	-	-	-	-	-	-
• LDR	1	F9	39	-	-	-	-	-	-
LDX	0	22	02	62	A2	42	82	-	-
LEA	0	-	04	64	A4	44	84	-	-
MUL	0	2C	0C	6C	AC	4C	8C	-	-
* MVS	0'	-	1F	7F	BF	5F	9F	-	-
• RD	1	F402	-	-	-	-	-	-	-
• RSV	1	F1	-	-	-	-	-	-	-
RTS	1	F100	-	-	-	-	-	-	-
SBL	0'	-	14	74	B4	54	94	-	-
SBR	0'	-	15	75	B5	55	95	-	-
* SHC	1	FC	3C	-	-	-	-	o EC	-
SHR	1	F0	30	-	-	-	-	o E0	-
SPA	0'	-	18	78	B8	58	98	-	-
SRG	1	F1	31	-	-	-	-	o E1	-
STA	0'	-	11	71	B1	51	91	-	-
STE	0'	-	12	72	B2	52	92	-	-
• STM	1	F408	-	-	-	-	-	-	-
• STR	1	FA	3A	-	-	-	-	o EA	-
STS	0'	-	19	79	B9	59	99	-	-
STX	0'	-	13	73	B3	53	93	-	-
SUB	0	26	06	66	A6	46	86	-	-
*TES	1	FD	3D	-	-	-	-	o ED	-
*TRS	0'	-	1E	7E	BE	5E	9E	-	-
• WD	1	F403	-	-	-	-	-	-	-

Nota :

• : instruction privilégiée

* : instruction optionnelle

o : adressage PX

Liste des instructions par code hexadécimal croissant

Code	Instruc.	Classe	Adr.	Code	Instruc.	Classe	Adr.	Code	Instruc.	Classe	Adr.	Code	Instruc.	Classe	Adr.
00	LDA	0	DL	20	LDA	0	P	40	LDA	0	DG	60	LDA	0	IL
01	LDE	0	-	21	LDE	0	-	41	LDE	0	-	61	LDE	0	-
02	LDX	0	-	22	LDX	0	-	42	LDX	0	-	62	LDX	0	-
03	EOR	0	-	23	EOR	0	-	43	EOR	0	-	63	EOR	0	-
04	LEA	0	-	24	LEA	0	-	44	LEA	0	-	64	LEA	0	-
05	ADD	0	-	25	ADD	0	-	45	ADD	0	-	65	ADD	0	-
06	SUB	0	-	26	SUB	0	-	46	SUB	0	-	66	SUB	0	-
07	IOR	0	-	27	IOR	0	-	47	IOR	0	-	67	IOR	0	-
08	* DIV	0	-	28	* DIV	0	-	48	* DIV	0	-	68	* DIV	0	-
09	AND	0	-	29	AND	0	-	49	AND	0	-	69	AND	0	-
0A	* CPS	0	-	2A	* CPS	0	-	4A	* CPS	0	-	6A	* CPS	0	-
0B	CMP	0	-	2B	CMP	0	-	4B	CMP	0	-	6B	CMP	0	-
0C	MUL	0	-	2C	MUL	0	-	4C	MUL	0	-	6C	MUL	0	-
0D	LBL	0	-	2D	LBL	0	-	4D	LBL	0	-	6D	LBL	0	-
0E	LBR	0	-	2E	LBR	0	-	4E	LBR	0	-	6E	LBR	0	-
0F	LBX	0	-	2F	LBX	0	-	4F	LBX	0	-	6F	LBX	0	-
10	DLD	0'	-	30	SHR	1	DL	50	DLD	0'	-	70	DLD	0'	-
11	STA	0'	-	31	SRG	1	-	51	STA	0'	-	71	STA	0'	-
12	STE	0'	-	32	ICX	1	-	52	STE	0'	-	72	STE	0'	-
13	STX	0'	-	33	DCX	1	-	53	STX	0'	-	73	STX	0'	-
14	SBL	0'	-	34				54	SBL	0'	-	74	SBL	0'	-
15	SBR	0'	-	35	ICL	1	-	55	SBR	0'	-	75	SBR	0'	-
16	DST	0'	-	36	DCL	1	-	56	DST	0'	-	76	DST	0'	-
17	ADM	0'	-	37	CSV	1	-	57	ADM	0'	-	77	ADM	0'	-
18	SPA	0'	-	38	CLS	1	-	58	SPA	0'	-	78	SPA	0'	-
19	STS	0'	-	39	• LDR	1	-	59	STS	0'	-	79	STS	0'	-
1A	* FAD	0'	-	3A	• STR	1	-	5A	* FAD	0'	-	7A	* FAD	0'	-
1B	* FSU	0'	-	3B	* • LDP	1	-	5B	* FSU	0'	-	7B	* FSU	0'	-
1C	* FMU	0'	-	3C	* SHC	1	-	5C	* FMU	0'	-	7C	* FMU	0'	-
1D	* FDV	0'	-	3D	* TES	1	-	5D	* FDV	0'	-	7D	* FDV	0'	-
1E	* TRS	0'	-	3E				5E	* TRS	0'	-	7E	* TRS	0'	-
1F	* MVS	0'	-	3F				5F	* MVS	0'	-	7F	* MVS	0'	-

Nota :

* : instruction en option
• : instruction privilégiée

Code	Instruc.	Classe	Adr.	Code	Instruc.	Classe	Adr.	Code	Instruc.	Classe	Adr.	Code	Instruc.	Classe	Adr.
80	LDA	0	IGX	A0	LDA	0	ILX	C0	BCT	2	RP	E0	SHR	1	PX
81	LDE	0	-	A1	LDE	0	-	C1	BRX	2	-	E1	SRG	1	-
82	LDX	0	-	A2	LDX	0	-	C2	BOT	2	-	E2	ICX	1	-
83	EOR	0	-	A3	EOR	0	-	C3	BCF	2	-	E3	DCX	1	-
84	LEA	0	-	A4	LEA	0	-	C4	BAN	2	-	E4			
85	ADD	0	-	A5	ADD	0	-	C5	BAZ	2	-	E5	ICL	1	-
86	SUB	0	-	A6	SUB	0	-	C6	BOF	2	-	E6	DCL	1	-
87	IOR	0	-	A7	IOR	0	-	C7	BRU	2	-	E7	CSV	1	-
88	*DIV	0	-	A8	*DIV	0	-	C8	BCT	2	RM	E8	CLS	1	-
89	AND	0	-	A9	AND	0	-	C9	BRX	2	-	E9	• LDR	1	-
8A	*CPS	0	-	AA	*CPS	0	-	CA	BOT	2	-	EA	• STR	1	-
8B	CMP	0	-	AB	CMP	0	-	CB	BCF	2	-	EB	*•LDP	1	-
8C	MUL	0	-	AC	MUL	0	-	CC	BAN	2	-	EC	* SHC	1	-
8D	LBL	0	-	AB	LBL	0	-	CD	BAZ	2	-	ED	* TES	1	-
8E	LBR	0	-	AE	LBR	0	-	CE	BOF	2	-	EE			
8F	LBX	0	-	AF	LBX	0	-	CF	BRU	2	-	EF			
90	DLD	0'	-	B0	DLD	0'	-	D0	BCT	2	IL	F0	SHR	1	P
91	STA	0'	-	B1	STA	0'	-	D1	BRX	2	-	F1	SRG	1	-
92	STE	0'	-	B2	STE	0'	-	D2	BOT	2	-	F2	ICX	1	-
93	STX	0'	-	B3	STX	0'	-	D3	BCF	2	-	F3	DCX	1	-
94	SBL	0'	-	B4	SBL	0'	-	D4	BAN	2	-	F4	• SYS(1)	1	-
95	SBR	0'	-	B5	SBR	0'	-	D5	BAZ	2	-	F5	ICL	1	-
96	DST	0'	-	B6	DST	0'	-	D6	BOF	2	-	F6	DCL	1	-
97	ADM	0'	-	B7	ADM	0'	-	D7	BRU	2	-	F7	CSV	1	-
98	SPA	0'	-	B8	SPA	0'	-	D8	BCT	2	IG	F8	CLS	1	-
99	STS	0'	-	B9	STS	0'	-	D9	BRX	2	-	F9	• LDR	1	-
9A	*FAD	0'	-	BA	*FAD	0'	-	DA	BOT	2	-	FA	• STR	1	-
9B	*FSU	0'	-	BB	*FSU	0'	-	DB	BCF	2	-	FB	*•LDP	1	-
9C	*FMU	0'	-	BC	*FMU	0'	-	DC	BAN	2	-	FC	* SHC	1	-
9D	*FDV	0'	-	BD	*FDV	0'	-	DD	BAZ	2	-	FD	* TES	1	-
9E	*TRS	0'	-	BE	*TRS	0'	-	DE	BOF	2	-	FE			
9F	*MVS	0'	-	8F	*MSV	0'	-	DF	BRU	2	-	FF			

Nota :

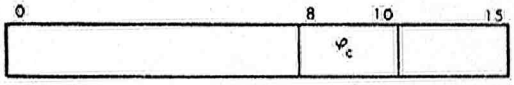
* : instruction en option
• : instruction privilégiée

(1) SYS : ce mnémonique n'est pas reconnu par l'assembleur

Instruction SRG

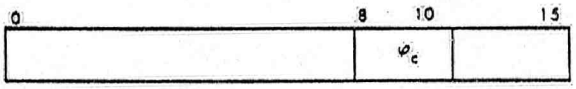
Instruction	Fonction	Code
AAE	(A) ∩ (E) ⟶ (A)	F118
ACE	(E) + C ⟶ (E)	F10E
AEE	(A) ⊕ (E) ⟶ (A)	F112
AIE	(A) ∪ (E) ⟶ (A)	F116
CCA	$(\overline{A})$ ⟶ (A)	F110
CCE	$(\overline{E})$ ⟶ (E)	F10A
CHX	Décalage arithmétique droit X 1 pas	F11E
CNA	-A ⟶ (A)	F11C
CNX	-X ⟶ (X)	F114
LNE	-1 ⟶ (E)	F11A
RTS	Retour section	F100
RSV	Retour superviseur	F10C
XAA	A_{0-7} ⟷ A_{8-15}	F108
XAE	(A) ⟷ (E)	F102
XAX	(A) ⟷ (X)	F104
XEX	(E) ⟷ (X)	F106

Instruction SHR



Fonction	φ_c	Code
Logique gauche A	0	SLLS
Circulaire droit A	1	SRCS
Arithmétique droit E,A	2	SAD
Circulaire gauche E,A	3	SLCD
Circulaire gauche A	4	SLCS
Arithmétique droit A	5	SAS
Logique droit A	6	SRLS
Circulaire droit E,A	7	SRCD

Instruction SHC



Fonction	φ_c	Code
Décalage logique gauche E,A	0	SLLD
Calcul de parité	2	PTY
Décalage logique droit E,A	4	SRLD
Normalisation E,A	6	NLZ
Désactivation de l'IT rapide	1 3 5 7	DITR

Instruction SYS[1]

Fonction	Classe	Instruction	Code
Démasquage IT	1	CLM	F400
Désactivation IT	1	DIT	F401
Lecture	1	RD	F402
Ecriture	1	WD	F403
Masquage IT	1	STM	F408

(1) SYS : ce mnémonique n'est pas reconnu par l'assembleur

Adressage :

Classe 0		Classe 0'		Classe 1		Classe 2	
P	Y = D	DL	Y = D + (L)	P	N = D	RP	Y = (P) + 2D
DL	Y = D + (L)	IL	Y = (D +(L))+G'	PX	N = D + (X)	RM	Y = (P) - 2D
IL	Y = (D +(L))+G'	ILX	Y = (D +(L))+G'+(X)	DL	N = (D + (L))	DL	Y = (D +(L))+G'
ILX	Y = (D +(L))+G'+(X)	DG	Y = D + (G)			DG	Y = (D +(G))+G'
DG	Y = D + (G)	IGX	Y = (D +(G))+(G)+(X)				
IGX	Y = (D +(G))+(G)+(X)						

annexe C_Modes d'adressage

1) Adressage classe 0

Les instructions de ce type peuvent adresser simplement toute donnée du segment local ou du segment commun. L'adressage de type 0' n'admet pas le mode paramètre.

Ce sont les instructions de :

- Chargement et rangement des registres
- Opérations arithmétiques (format fixe ou flottant)
- Opérations logiques
- Traitement de chaîne de caractères
- Comparaison

Elles admettent six modes d'adressage :

Mode d'adressage	Ecriture en Assembleur	Opérande	Fonction d'adressage
Direct local DL	IDENT	Octet, mot ou double mot situé dans les 256 premiers octets du segment local.	Y=(L)+D
Indirect local IL	@IDENT	Octet, mot ou double mot situé à un emplacement quelconque et pointé à travers le segment local.	Y=(G')+((L)+D)
Indirect local indexé ILX	@IDENT, X	Elément d'un tableau d'octets, de mots ou de double-mots situé à un emplacement quelconque et pointé à travers le segment local.	Y=(G')+((L)+D)+(X)
Direct général DG	#IDENT	Octet, mot ou double-mot situé dans les 256 premiers octets du segment commun.	Y=(G)+D
Indirect Général Indexé IGX	@#IDENT, X	Elément d'un tableau pointé à travers le segment commun.	Y=(G)+((G)+D)+(X)
Paramètre ou immédiat P	= PARAMETRE	L'opérande de 8 bits figure dans l'instruction. Ces 8 bits sont éventuellement étendus par 8 zéros en poids forts.	Y = D

2) Adressage classe 1

Les instructions de ce type sont :

- soit des instructions sans opérande : échange de registres, fin de section, etc...

- soit des instructions dont l'opérande est le plus souvent connu (ou connu à un modificateur près) lors de l'écriture du programme : décalage, incrément, index, etc...

- Ce sont les instructions de :

. décalages
. opérations sur index
. opérations sur bases
. appels section ou superviseur
. entrées/sorties
. opérations sur registre
. opérations sur interruptions et masque d'interruptions.

Ces instructions admettent trois modes d'adressages :

Mode d'adressage	Ecriture en Assembleur	Opérande	Fonction d'adressage
Paramètre P	= PARAM	L'opérande est défini par la valeur du déplacement.	y = D
Paramètre indexé PX	= PARAM, X	L'opérande est défini par la valeur du déplacement augmenté du contenu de X.	y = D + (X)
Direct local DL	IDENT	L'opérande est placé dans les 256 premiers octets du segment local.	Y = (L) + D

3) Adressage classe 2

Ce sont les instructions de branchement conditionnel ou inconditionnel.

Les instructions désignées par une instruction de rupture de séquence appartiennent normalement à la même section de programme que cette rupture.

Ces instructions admettent quatre modes d'adressages :

Mode d'adressage	Ecriture en Assembleur	Instruction de Branchement	Fonction d'adressage
Relatif plus RP	LABEL	Toute instruction a moins de 512 octets en aval.	(P)=(P)+2 D
Relatif moins RM	LABEL	Toute instruction a moins de 512 octets en amont.	(P)=(P)-2 D
Indirect local IL	@ LABEL	Toute instruction pointée à travers le segment local.	(P)=(G')+((L)+ D)
Indirect général IG	@#LABEL	Toute instruction pointée à travers le segment commun	(P)=(G')+(G)+ D

annexe D. Mise en œuvre de l'assembleur

I - LISTE D'ASSEMBLAGE

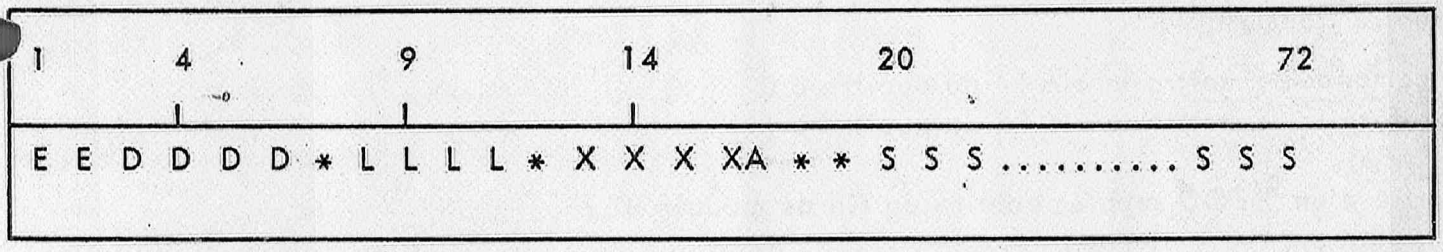
I-1. FORMAT D'UNE LIGNE DE LA LISTE D'ASSEMBLAGE

Le format général d'une ligne d'une liste d'assemblage est donné sur la figure ci-dessous.

Une ligne de liste contient les informations suivantes :

- Jusqu'à deux drapeaux d'erreur.
- Le numéro décimal de la ligne.
- Le contenu courant du compteur d'emplacement (hexadécimal).
- Le code objet produit par l'assembleur (hexadécimal).
- L'indication de référence en avant si l'adresse argument est anticipée.
- L'image de la ligne du programme-source.

Les lignes sautées sous contrôle de GOTO ne sont pas annotées, à l'exception des codes opérations interdits.



*	:	Espace
EE	:	Drapeaux d'erreur
DDDD	:	Numéro décimal de la ligne source
LLLL	:	Valeur hexadécimale du compteur d'emplacement
XXXX ou **XX	:	Valeur hexadécimale du mot ou de l'octet généré à l'emplacement LLLL
A	:	Sa présence indique une référence en avant
SS..SS	:	Ligne du texte source

Format d'une ligne de la liste d'assemblage

I-2. LISTE OBJET

En plus de la représentation hexadécimale du code objet et l'édition du texte source correspondant, l'assembleur fournit :

- La liste des noms de segments définis et/ou référencés excepté dans MITRAS 1
- La table des étiquettes satisfaites ou non satisfaites par section (option).

II - OPTIONS DE MISE EN OEUVRE

II-1. MITRAS 1

■ Format de la commande

%ASS 1/ options indiquées aux clés du pupitre (M:OC)

■ Options

Liste d'assemblage :

- demandée : mettre la clé 15 du pupitre à 0
- refusée : mettre la clé 15 du pupitre à 1

Binaire translatable :

- demandé : mettre la clé 14 du pupitre à 0
- refusé : mettre la clé 14 du pupitre à 1

Sortie d'un %EOD supplémentaire en fin de module BT :

- demandé : mettre la clé 13 du pupitre à 0
- refusé : mettre la clé 13 du pupitre à 1

II-2. MITRAS II

■ Format de la commande

{ % ASS 2 / % CALL/ASS 2 } / [SI] [,BO] [,LO] [,LL]

% ASS 2/ Sorti sur M:OC par le processeur lorsqu'il est chargé et lancé par %LOAD et %RUN

%CALL/ASS 2/ Sorti su M:OC par le processeur lorsqu'il est lancé sans interruptions pupitre. Dans ce cas les options sont données sur M:OC.

Sous le contrôle du module d'enchainement, la commande avec ses options est entrée sur M:CI

■ Options

- SI : Indique que le fichier source est lu sur M:SI. Lorsque cette option est absente, aucune autre ne doit figurer, et toutes les options sont implicites.
- BO : Indique que la sortie de binaire translatable est demandée.
- LO : Indique qua la sortie de liste d'assemblage est demandée.
- LL : Indique que l'impression des tables d'étiquettes ainsi que cette du niveau de sévérité et du nombre de lignes erronées est demandée.

• Option fin d'assemblage

Dans le cas où MITRAS 2 est chargé et lancé par %LOAD et %RUN

Avant qu'il ne reboucle à son début en fin d'assemblage, MITRAS 2 sort sur l'étiquette opérationnelle M:OC un message de la forme :

%%EOD ?

Ceci signifie que la possibilité de sortir une marque fin de fichier supplémentaire en fin de module BT est affecté.

L'opérateur répond alors sur l'étiquette opérationnelle M:OC :

- OUI s'il désire une marque fin de fichier supplémentaire.
- NON s'il ne désire pas de marque fin de fichier supplémentaire.

• Marque fin de fichier supplémentaire

Lorsqu'un module de programme est assemblé, le binaire translatable sorti peut être d'une des deux formes suivantes :

En-tête	Fichier	M F F

ou

En-tête	Fichier	M F F	M F F

- où M F F signifie "marque fin de fichier".

Ceci est nécessaire afin de pouvoir constituer le fichier BT d'entrée pour l'éditeur de liens. Ce fichier est de forme :

En-tête 1	Fichier 1	M F F	En-tête 2	Fichier 2	M F F	

En-tête n	Fichier n	M F F	M F F

L'éditeur de liens pourra ainsi sortir un fichier en IMT de la forme :

En-tête	Fichier	M F F	M F F

III-2. FORMAT DES TABLES

III-2.1. Table des étiquettes locales

Une table contenant la liste des étiquettes locales est sortie après chaque LPS avant la directive FIN. Son format est le suivant :

ETIQ LLLL Z

- ETIQ : Nom de l'étiquette
- LLLL : Valeur du compteur d'emplacement correspondant à cette étiquette; ou numéro de référence de l'étiquette dans le cas où celle-ci n'est pas défini.
- Z : D = Etiquette de LDS
  P = Etiquette de LPS
  A = La valeur de l'étiquette est en absolu
  X = L'étiquette n'est pas définie
  R = L'étiquette figure dans une directive REF

III-2.2. Table des étiquettes communes

A la fin d'un module, après la directive END, une table contenant les étiquettes communes est sortie. Son format est le suivant :

ETIQ LLLL Z

- ETIQ : Nom de l'étiquette
- LLLL : Valeur du compteur d'emplacement correspondant à cette étiquette; ou numéro de référence de l'étiquette dans le cas où celle-ci n'est pas définie.
- Z : C = Etiquette de CDS
  : D = Etiquette de LDS (figurant dans une directive DEF en CDS et définie en LDS).
  : P = Etiquette de LPS (figurant dans une directive DEF en CDS et définie en LPS).
  : A = La valeur de l'étiquette est en absolue.
  : X = L'étiquette n'est pas définie.
  : R = L'étiquette figure dans une directive REF