NORME INTERNATIONALE

INTERNATIONAL STANDARD

CEI
IEC
60975

Première édition
First edition
1989-08

Format universel pour les modules objets de microprocesseurs

Microprocessor universal format for object modules

Numéro de référence
Reference number
CEI/IEC 60975: 1989

Numéros des publications

Depuis le 1er janvier 1997, les publications de la CEI sont numérotées à partir de 60000.

Publications consolidées

Les versions consolidées de certaines publications de la CEI incorporant les amendements sont disponibles. Par exemple, les numéros d'édition 1.0, 1.1 et 1.2 indiquent respectivement la publication de base, la publication de base incorporant l'amendement 1, et la publication de base incorporant les amendements 1 et 2.

Validité de la présente publication

Le contenu technique des publications de la CEI est constamment revu par la CEI afin qu'il reflète l'état actuel de la technique.

Des renseignements relatifs à la date de reconfirmation de la publication sont disponibles dans le Catalogue de la CEI.

Les renseignements relatifs à des questions à l'étude et des travaux en cours entrepris par le comité technique qui a établi cette publication, ainsi que la liste des publications établies, se trouvent dans les documents ci-dessous:

- **«Site web» de la CEI***
- **Catalogue des publications de la CEI**
  Publié annuellement et mis à jour régulièrement
  (Catalogue en ligne)*
- **Bulletin de la CEI**
  Disponible à la fois au «site web» de la CEI* et comme périodique imprimé

Terminologie, symboles graphiques et littéraux

En ce qui concerne la terminologie générale, le lecteur se reportera à la CEI 60050: *Vocabulaire Electrotechnique International* (VEI).

Pour les symboles graphiques, les symboles littéraux et les signes d'usage général approuvés par la CEI, le lecteur consultera la CEI 60027: *Symboles littéraux à utiliser en électrotechnique,* la CEI 60417: *Symboles graphiques utilisables sur le matériel. Index, relevé et compilation des feuilles individuelles,* et la CEI 60617: *Symboles graphiques pour schémas.*

* Voir adresse «site web» sur la page de titre.

Numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series.

Consolidated publications

Consolidated versions of some IEC publications including amendments are available. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Validity of this publication

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology.

Information relating to the date of the reconfirmation of the publication is available in the IEC catalogue.

Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is to be found at the following IEC sources:

- **IEC web site***
- **Catalogue of IEC publications**
  Published yearly with regular updates
  (On-line catalogue)*
- **IEC Bulletin**
  Available both at the IEC web site* and as a printed periodical

Terminology, graphical and letter symbols

For general terminology, readers are referred to IEC 60050: *International Electrotechnical Vocabulary* (IEV).

For graphical symbols, and letter symbols and signs approved by the IEC for general use, readers are referred to publications IEC 60027: *Letter symbols to be used in electrical technology*, IEC 60417: *Graphical symbols for use on equipment. Index, survey and compilation of the single sheets* and IEC 60617: *Graphical symbols for diagrams.*

* See web site address on title page.

NORME INTERNATIONALE

INTERNATIONAL STANDARD

CEI
IEC
60975

Première édition
First edition
1989-08

Format universel pour les modules objets de microprocesseurs

Microprocessor universal format for object modules

International Electrotechnical Commission 3, rue de Varembé Geneva, Switzerland
Telefax: +41 22 919 0300 e-mail: inmail@iec.ch IEC web site http: //www.iec.ch

IEC Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

CODE PRIX
PRICE CODE W

Pour prix, voir catalogue en vigueur
For price, see current catalogue

SOMMAIRE

CONTENTS

Section Pages

Section Page

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

FORMAT UNIVERSEL POUR LES MODULES OBJETS DE MICROPROCESSEURS

PREAMBULE

1) Les décisions ou accords officiels de la CEI en ce qui concerne les questions techniques, préparés par des Comités d'Etudes où sont représentés tous les Comités nationaux s'intéressant à ces questions, expriment dans la plus grande mesure possible un accord international sur les sujets examinés.

2) Ces décisions constituent des recommandations internationales et sont agréées comme telles par les Comités nationaux.

3) Dans le but d'encourager l'unification internationale, la CEI exprime le voeu que tous les Comités nationaux adoptent dans leurs règles nationales le texte de la recommandation de la CEI, dans la mesure où les conditions nationales le permettent. Toute divergence entre la recommandation de la CEI et la règle nationale correspondante doit, dans la mesure du possible, être indiquée en termes clairs dans cette dernière.

PREFACE

La présente norme a été établie par le Sous-Comité 47B*: Systèmes à microprocesseurs, du Comité d'Etudes n° 47 de la CEI: Dispositifs à semiconducteurs.

La présente norme a été reprise de la Norme 695 de l'IEEE: IEEE Trial-use Standard for Microprocessor Universal Format for Object Modules (1985).

Le texte de cette norme est issu des documents suivants:

Règle des Six Mois	Rapport de vote
47B(BC)24	47B(BC)30

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

---

* Le Sous-Comité 47B de la CEI est désormais transféré dans l'ISO/CEI JTC 1.

La présente norme ayant été approuvée selon les procédures de la CEI, elle est par conséquent publiée comme norme de la CEI.

INTERNATIONAL ELECTROTECHNICAL COMMISSION

MICROPROCESSOR UNIVERSAL FORMAT FOR OBJECT MODULES

FOREWORD

1) The formal decisions or agreements of the IEC on technical matters, prepared by Technical Committees on which all the National Committees having a special interest therein are represented, express, as nearly as possible, an international consensus of opinion on the subjects dealt with.

2) They have the form of recommendations for international use and they are accepted by the National Committees in that sense.

3) In order to promote international unification, the IEC expresses the wish that all National Committees should adopt the text of the IEC recommendation for their national rules in so far as national conditions will permit. Any divergence between the IEC recommendation and the corresponding national rules should, as far as possible, be clearly indicated in the latter.

PREFACE

This standard has been prepared by Sub-Committee 47B*: Microprocessor systems, of IEC Technical Committee No. 47: Semiconductor devices.

This standard is based on IEEE Standard 695: IEEE Trial-Use Standard for Microprocessor Universal Format for Object Modules (1985).

The text of this standard is based upon the following documents:

Six Months' Rule	Report on Voting
47B(CO)24	47B(CO)30

Full information on the voting for the approval of this standard can be found in the Voting Report indicated in the above table.

* IEC Sub-Committee 47B has now been transferred to ISO/IEC JTC 1.

This standard was approved according to IEC procedures and is therefore published as an IEC standard.

FORMAT UNIVERSEL POUR LES MODULES OBJETS DE MICROPROCESSEURS

1. Domaine d'application

La présente norme spécifie le format de modules objets reliables, relogeables et absolus destinés à des ordinateurs de tailles de mots et d'architecture arbitraires. Deux niveaux de conformité sont spécifiés, minimal et total.

Le niveau de conformité minimal permet une souplesse suffisante pour relier des modules compilés séparément, pour translater des adresses de manière simple et pour charger les modules objets absolus résultants avec un chargeur minimal.

Le niveau de conformité total permet toutes les fonctionnalités du niveau minimal et lui adjoint la gestion d'expressions arbitraires d'adresses, la possibilité de contrôle de type, des commandes de gestion de bibliothèques, et d'autres fonctions utiles pour une généralité complète.

Une réalisation conforme peut étendre l'ensemble des commandes ou des fonctions de FUMOM en vue d'une meilleure efficacité pour traiter les exigences spécifiques à la machine, mais les modules objets contenant de telles extensions ne doivent alors pas être réputés conformes à cette norme. De telles extensions ne sont pas spécifiées dans cette norme.

Cette norme ne spécifie pas non plus:

1) Les caractéristiques du langage source devant être supportées grâce aux modules FUMOM.
2) La structure des librairies de code relogeable.
3) Le format de l'équivalent binaire optionnel.
4) Les algorithmes pour l'édition de liens, la translation d'adresses et le chargement.
5) L'architecture de la machine hôte.
6) L'architecture de la machine cible, y compris la taille des mots, la taille de la mémoire, l'organisation de la mémoire ou le format des instructions.

Cette norme est issue de documents du domaine public et ne contient aucun élément breveté ou d'usage privé.

2. Définitions

Les définitions suivantes donnent la signification des mots techniques tels qu'ils sont utilisés dans cette norme et servent à traduire l'acception usuelle des mots définis:

code absolu: Donnée ou code machine exécutable en mémoire ou son image. A distinguer de code relogeable.

MICROPROCESSOR UNIVERSAL FORMAT FOR OBJECT MODULES

1. Scope

This standard specifies the format of linkable, relocatable, and absolute object modules for binary computers of arbitrary word size and architecture. Two levels of compliance are specified, minimum and full.

The minimum compliance level affords sufficient flexibility to link separately compiled modules, to relocate addresses in simple ways, and to load the resulting absolute object modules with a minimal loader.

The full compliance level affords all of the functionality of the minimum level and adds to it arbitrary address expression handling, type checking capability, librarian control commands, and other useful functions for full generality.

A conforming implementation may extend the command or function set of MUFOM for greater efficiency in dealing with machine-specific requirements, but object modules containing such extensions shall not be said to be conforming to this standard. Such extensions are not specified in this standard.

This standard also does not specify:

1) The source language features to be supported by way of MUFOM modules.
2) The structure of relocatable code libraries.
3) The format of the optional binary equivalent.
4) Algorithms for linking, relocating and loading.
5) The architecture of the host machine.
6) The architecture of the target machine, including word size, size of memory, memory organization or instruction format.

This standard is derived from documents in the public domain and contains no patented or proprietary material.

2. Definitions

The following definitions give the meanings of the technical words as they are used in this standard and are intended to reflect the accepted use of the defined words:

absolute code: Data or executable machine code in memory or an image thereof. Distinguish from relocatable code.

chargeur absolu: Un processus pouvant charger une ou plusieurs sections de code absolu seulement aux adresses spécifiées par les sections. Voir: relogeur.

information binaire: Configuration de bits devant être chargée en mémoire.

forme caractère: La représentation sous forme de caractère (imprimable) de l'information binaire par opposition à la représentation sous forme de configuration de bits.

somme de contrôle: Fonction déterministe du contenu d'un fichier. Si un fichier est copié et que la somme de contrôle de la copie est différente de celui de l'original, il y a eu une erreur lors de l'opération de copie.

code: Donnée ou code machine exécutable. Voir: code absolu et code relogeable.

commande: Information de commande d'un éditeur de liens, d'un relogeur ou d'un chargeur. A distinguer de code.

commentaires: Information lisible par l'homme. A distinguer d'information binaire et de forme caractère.

définition externe: Définition, à l'intérieur d'une section, d'un symbole auquel il est fait référence dans d'autres sections. Un symbole global exporté.

référence externe: Spécification, dans une section, d'un symbole qui est défini en dehors de cette section. Un symbole global importé.

format: Le langage dans lequel sont spécifiés les modules objets.

bibliothécaire: Processus qui exécute des opérations, telles que la maintenance, sur une bibliothèque.

bibliothèque: Ensemble de modules objets.

éditeur de liens: Processus qui combine plusieurs modules objets en un seul module objet satisfaisant aux liens entre ceux-ci.

pointeur de chargement: Pointeur relatif à une section géré dynamiquement par le chargeur. Il indique à quel endroit l'élément de code suivant doit être chargé. Il est initialisé à une adresse de chargement de départ.

chargeur: Synonyme de chargeur absolu.

UMA: Abréviation de "unité minimale adressable".

unité minimale adressable: Représente, pour un processeur donné, la mémoire comprise entre une adresse et la suivante. Elle n'est pas nécessairement équivalente à un mot ou à un octet. Abréviation: UMA.

module: Synonyme de module objet.

absolute loader: A process which can load one or more sections of absolute code only at the locations specified by the sections. See: relocator.

binary information: Bit patterns to be loaded into memory.

character form: The (printable) character representation of binary information as opposed to the bit pattern representation.

checksum: A deterministic function of a file's contents. If a file is copied and the checksum of the copy is different from the original, there has been an error in copying.

code: Data or executable machine code. See: absolute code and relocatable code.

command: Control information for a linker, relocator, or loader. Distinguish from code.

comments: Information that is readable by people. To be distinguished from binary information and character form.

external definition: The definition, within a section, of a symbol which is referenced by other sections. An exported global symbol.

external reference: The specification, within a section, of a symbol which is defined outside that section. An imported global symbol.

format: The language in which object modules are specified.

librarian: The process which performs operations, such as maintenance, upon a library.

library: A set of object modules.

linkage editor (or linker): A process which combines object modules into a single object module satisfying links between the object modules.

load pointer: A pointer for a section which is dynamically maintained by the loader. It indicates where the next item of code is to be loaded. It is initialized to a starting load address.

loader: Same as absolute loader.

MAU: Abbreviation for minimum addressable unit.

minimum addressable unit: For a given processor, the amount of memory located between an address and the next address. It is not necessarily equivalent to a word or a byte. Abbreviated MAU.

module: Same as object module.

FUMOM: Acronyme de Format Universel pour les Modules Objets de Microprocesseurs.

module objet: Un ensemble de sections de code absolu ou relogeable, associé à des commandes.

processus: Un programme qui est exécuté par un processeur. Dans cette norme, la dénotation de ce terme est restreinte à éditeur de liens, relogeur, évaluateur d'expressions et chargeur.

processeur: Une machine de calcul, quoique peut-être une machine virtuelle.

programme: Un ensemble de données et d'instructions qui définit les opérations à effectuer par un processeur.

code relogeable: Code indépendant de son emplacement qui peut être chargé à une adresse arbitraire de la mémoire. Il requiert généralement à la fois le relogement et l'évaluation d'expressions pour devenir du code absolu.

relogeur: Un processus qui affecte des adresses absolues à une section de code relogeable en vue de produire du code absolu.

relogement: Fonction du relogeur.

copie multiple: Le chargement de données répétitives grâce à une spécification abrégée.

section: Partie d'un programme dotée d'information auxiliaire (commandes) qui devient un segment lorsqu'elle est chargée.

segment: Région d'un seul tenant de la mémoire, dotée de frontières arbitraires.

doit: Est requis de..., sont requis de...

devrait: Est conseillé de..., sont conseillés de...

adresse de chargement de départ: L'adresse à laquelle le chargement d'une section commence.

type: Attribut d'un symbole, d'une valeur ou d'une section. La définition de type pour un symbole est laissée au réalisateur.

référence externe faible: Référence externe qui ne doit pas nécessairement être satisfaite lors de l'édition de liens.

3. Références

[1] ANSI X3.4-1977: American National Standard Code for Information Interchange.

[2] ANSI X3.43-1977: American National Standard Representations of Local Time of the Day for Information Interchange.

[3] ISO 3307-1975: Echange d'information - Représentations de l'heure.

MUFOM: Acronym for Microprocessor Universal Format for Object Modules.

object module : A set of sections of absolute code or relocatable code, together with commands.

process: A program which is executed by a processor. In this standard, the denotation is restricted to linker, relocator, expression evaluator and loader.

processor: A computing machine, albeit perhaps a virtual machine.

program: A set of data and instructions which defines the operations to be performed by a processor.

relocatable code: Position-independent code which can be loaded at an arbitrary memory location. It generally requires both relocation and expression evaluation to become absolute code.

relocator: A process which assigns absolute addresses to a section of relocatable code to produce absolute code.

relocation: The function of the relocator.

replication: The loading of repetitive data by means of an abbreviated specification.

section: A part of a program with ancillary information (commands) which becomes a segment when loaded.

segment: A contiguous region in memory with arbitrary boundaries.

shall: Is required to..., are required to...

should: Is advised to..., are advised to...

starting load address: The address at which the loading of a section begins.

type: An attribute of a symbol, value or section. The definition of a symbol type is left to the implementer.

weak external: An external reference which need not be satisfied during linking.

3. References

[1] ANSI X3.4-1977: American National Standard Code for Information Interchange.

[2] ANSI X3.43-1977: American National Standard Representations of Local Time of the Day for Information Interchange.

[3] ISO 3307-1975: Representations of Time of the Day - Information Interchange.

4. Introduction

4.1 *Commandes*

Un module objet FUMOM est une séquence de commandes. Ces commandes sont introduites grâce à un mnémonique de deux lettres et sont terminées par un point. Ainsi,

```
MBI8080.
ASP,100.
LDC30001.
ME.
```

est un module bien formé. Les descriptions détaillées des commandes de cet exemple sont contenues dans les sections suivantes, mais une paraphrase informelle de cet exemple est la suivante:

Le module commence "Intel 8080."

Affecter la valeur 256 (décimal) au pointeur de chargement.

Charger trois octets avec l'information binaire représentée par la spécification hexadécimale "C3," "00," et "01."

Fin du module

4.2 *Syntaxe en format libre*

Un module objet est traité comme une chaîne de caractères ASCII contigus conformément à la norme ANSI X3.4 [1]*. Tous les caractères ASCII de contrôle (tels que changement de ligne ou retour chariot) sont ignorés lors du traitement.

La longueur d'une commande FUMOM est variable et il n'y a aucune limitation à cette longueur. Pour représenter un module objet FUMOM comme un fichier texte sur une machine hôte (ayant une longueur de ligne limitée), une commande simple peut occuper plusieurs lignes successives (elle peut, par exemple, comprendre plusieurs changements de ligne, retours chariot ou d'autres caractères de contrôle).

5. Constructions de base

5.1 *Notation utilisée pour les définitions de la syntaxe*

Le métalangage utilisé pour les définitions de la syntaxe est dérivé de la forme de Backus-Naur et de la notation concernant les expressions régulières.

Les symboles terminaux (caractères littéraux indiqués exactement comme ils sont utilisés) sont inclus entre apostrophes; les symboles non terminaux (noms des formes syntaxiques) sont des mots (éventuellement munis de tirets) sans apostrophes. Des parenthèses sont utilisées pour indiquer les groupements. Les espaces ne sont pas significatifs. La concaténation s'indique par la juxtaposition des symboles. La répétition s'indique par la récursion ou par les deux métasymboles + ou *

---

* Les nombres entre crochets correspondent aux références indiquées dans la section 3.

4. Introduction

4.1 *Commands*

A MUFOM object module is a sequence of commands. These commands are introduced by a two-letter mnemonic and are terminated by a period. Thus,

MB18080.
ASP,100.
LDC30001.
ME.

is a well-formed module. Detailed descriptions of the commands in this example are contained in following sections, but an informal paraphrase of this example is:

Module begin "Intel 8080."

Assign load pointer value to be 256 (decimal).

Load three bytes with the binary information represented by the hexadecimal specification "C3," "00," and "01."

Module end.

4.2 *Free-form syntax*

An object module is treated as one contiguous string of ASCII characters according to ANSI standard X3.4 [1]*. All ASCII control characters (such as a line feed or a carriage return) are ignored in processing.

The length of a MUFOM command is variable and there is no limitation on that length. To represent a MUFOM object module as a text file on a host processor (with limited line size), a single command can occupy several successive lines (for example, encompass several line feeds, carriage returns, or other control characters).

5. Basic constructions

5.1 *Notation used for syntax definitions*

The metalanguage for the syntax definitions is derived from the Backus-Naur form and from regular expression notation.

Terminal symbols (literal characters shown exactly as used) are enclosed in quotation marks; nonterminal symbols (names of syntactic forms) are words (possibly hyphenated) without quotation marks. Parentheses are used to show grouping. Spaces are not significant. Concatenation is indicated by the juxtaposition of symbols. Repetition is indicated by recursion or by the two metasymbols + or *

---

* The numbers in square brackets correspond to the references listed in Section 3.

élément * = zéro ou plusieurs occurrences de élément

élément + = une ou plusieurs occurrences de élément

L'alternative est indiquée par le métasymbole | élément 1 | élément 2: soit élément 1 soit élément 2, exclusivement
L'option est indiquée par le métasymbole ?

, élément? : soit élément soit rien

EXEMPLE:

variable → lettre (lettre | digit)*

5.2 *Objets syntaxiques élémentaires*

FUMOM utilise un petit nombre de types de données pour fournir les moyens habituels pour exprimer les constructions. Ce sont les suivants:

digit → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

lettre-hexadécimale → "A" | "B" | "C" | "D" | "E" | "F"

digit-hexadécimal → digit | lettre-hexadécimale

nombre-hexadécimal → digit-hexadécimal +

lettre-non-hexadécimale → "G" | "H" | "I" | "J" | "K" | "L" |
"M" | "N" | "O" | "P" | "Q" | "R" |
"S" | "T" | "U" | "V" | "W" | "X" |
"Y" | "Z"

lettre → lettre-hexadécimale | lettre-non-hexadécimale

alphanumérique → lettre | digit

identificateur → lettre alphanumérique*

caractère → tout caractère ASCII graphique

longueur-chaîne-caractère → digit-hexadécimal digit-hexadécimal

chaîne-caractère → longueur-chaîne-caractère caractère*

où la valeur hexadécimale de longueur-chaîne-caractère est prise égale au nombre de caractères, qui doit être inférieur à 128, par exemple 14A STRING WITH SPACES.

variable-FUMOM → lettre-non-hexadécimale nombre-hexadécimal?

un nom qui fait référence à une variable interne d'un processus FUMOM (voir section 7).

item * = zero or more occurrences of item

item + = one or more occurrences of item

Alternation is indicated by the metasymbol | item 1 | item 2: either item 1 or item 2 not both
Election is indicated by the metasymbol ?

, item? : either item or nothing

EXAMPLE:

variable → letter (letter | digit)*

5.2 *Elementary syntactic objects*

MUFOM uses a small number of data types to provide standard means of expressing constructs. They are as follows:

digit → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

hexletter → "A" | "B" | "C" | "D" | "E" | "F"

hexdigit → digit | hexletter

hexnumber → hexdigit +

nonhexletter → "G" | "H" | "I" | "J" | "K" | "L" |
"M" | "N" | "O" | "P" | "Q" | "R" |
"S" | "T" | "U" | "V" | "W" | "X" |
"Y" | "Z"

letter → hexletter | nonhexletter

alphanum → letter | digit

identifier → letter alphanum*

character → any ASCII graphic character

char-string-length → hexdigit hexdigit

char-string → char-string-length character*

Where the hexadecimal value of string-length shall be equal to the number of characters, which shall be less than 128, for example 14A STRING WITH SPACES,

MUFOM-variable → nonhexletter hexnumber?

a name which references an internal variable of a MUFOM process (see Section 7).

6. Expressions

Les expressions sont écrites en notation Polonaise postfixée. La représentation binaire de chaque opérande ne doit pas excéder 64 bits. Une réalisation doit accepter des opérandes d'au moins 15 bits.

expression → nombre-hexadécimal | variable-FUMOM | expression-polonaise | expression-conditionnelle

expression-polonaise → (opérande ",")* fonction

opérande → expression

fonction → "@" identificateur | opérateur

opérateur → "+" | "-" | "/" | "*" | "<" | ">" | "=" | "#"

Lorsqu'une fonction ou un opérateur apparaît dans une expression, il spécifie le calcul à effectuer sur les valeurs données dans toute expression antérieure selon la définition de la fonction. Quelques fonctions standard sont fournies avec FUMOM. Cependant, cet ensemble sera vraisemblablement amélioré par le réalisateur de manière à plus aisément s'adapter à sa machine cible.

6.1 *Types d'opérandes*

Le résultat d'une opération effectuée sur les opérandes dépend des types et des valeurs de ces opérandes. Si le type ou le domaine d'un opérande n'est pas permis par l'opération qui lui est appliquée, le résultat est indéfini. Dans FUMOM, on distingue seulement deux types d'opérandes: logique et entier. Les opérandes entiers peuvent être positifs ou négatifs. Les opérandes logiques peuvent avoir seulement les deux valeurs VRAI et FAUX.

Le type d'un opérande écrit comme 'nombre-hexadécimal' est le type entier, et sa valeur est égale à celle du 'nombre-hexadécimal' considéré comme un nombre hexadécimal positif. Le type d'un opérande écrit comme une 'variable-FUMOM' dépend de la dernière affectation effectuée à la variable correspondante. Le type d'un opérande écrit comme une 'expression-polonaise' est défini ci-dessous. Le type d'un opérande écrit comme une 'expression-conditionnelle' est le type de l'opérande pour lequel la condition est vraie.

Pour les opérations logiques au niveau du bit (@AND, @OR, @XOR, @INS, @EXT), un entier FUMOM est représenté par un nombre binaire aligné sur la droite, complété par des zéros ou tronqué à gauche, et considéré comme non signé.

6.2 *Fonctions sans opérandes*

expression-polonaise → "@F"

expression-polonaise → "@T"

@F, @T - Ces fonctions renvoient les valeurs logiques FAUX et VRAI. Elles n'ont pas d'opérandes.

6. Expressions

Expressions are written in postfix Polish notation. The binary representation of each operand shall not exceed 64 bits. An implementation shall accept operands of at least 15 bits.

expression → hexnumber | MUFOM-variable | polish-expr | conditional-expr

polish-expr → (operand ",")* function

operand → expression

function → "@" identifier | operator

operator → "+" | "-" | "/" | "*" | "<" | ">" | "=" | "#"

When a function or an operator appears in an expression, it specifies the computation to be performed on the values given in any preceding expression or expressions according to the function definition. Some standard functions are provided with MUFOM. However, it is expected that the implementer will enhance this set in order to more easily accommodate his target machine.

6.1 *Types of operands*

The result of an operation performed on operands depends on the types and values of those operands. If the type or range of an operand is not allowed by the operation applied to it, the result is undefined. In MUFOM, only two types of operands are distinguished: logical and integer. Integer operands can be positive or negative. Logical operands can have only the two values TRUE and FALSE.

The type of an operand written as 'hexnumber' is the integer, and its value is equal to that of the 'hexnumber' considered as a positive hexadecimal number. The type of an operand written as 'MUFOM-variable' depends on the last assignment to the corresponding variable. The type of an operand written as 'polish-expr' will be defined below. The type of an operand written as 'conditional-expr' is the type of the operand for which the condition is true.

For bit-wise logical operations (@AND, @OR, @XOR, @INS, @EXT), a MUFOM integer is represented by a binary number aligned on the right, zero-filled or truncated on the left, and considered as unsigned.

6.2 *Functions with no operands*

polish-expr → "@F"

polish-expr → "@T"

@F, @T - These functions return the logical values FALSE and TRUE. They have no operands.

6.3 *Opérateurs monadiques et fonctions*

expression-polonaise → opérande "," f-monad

f-monad → "@ABS" | "@NEG" | "@NOT" | "@ISDEF"

@ABS - Le type de 'opérande' doit être entier. Cette fonction retourne un entier qui est la valeur absolue de 'opérande'.

@NEG - Le type de 'opérande' doit être entier. Cette fonction retourne un entier qui vaut le complément arithmétique de 'opérande'.

@NOT - Si le type de 'opérande' est logique, la valeur retournée est son complément logique. Si le type de 'opérande' est entier, la valeur retournée est le complément à un de sa valeur.

@ISDEF - Cette fonction retourne la valeur logique VRAI si 'opérande' ne contient aucune variable non affectée, et retourne FAUX dans le cas contraire.

6.4 *Opérateurs et fonctions dyadiques*

expression-polonaise → opérande1 "," opérande2 "," f-dyad

f-dyad → "+" | "-" | "/" | "*" | "@MAX" | "@MIN" | "@MOD" |
"<" | ">" | "=" | "#" | "@AND" | "@OR" | "@XOR"

Pour les fonctions non commutatives, l'ordre conventionnel des opérandes est utilisé. Par exemple, dans le cas de la division, le résultat est la valeur de opérande1 divisée par la valeur de opérande2.

+, -, /, * - Ces opérateurs exécutent les opérations arithmétiques habituelles sur deux opérandes, qui doivent être entiers. Le résultat est entier. La division tronque vers zéro et son résultat est indéfini si opérande2 est nul.

@MAX, @MIN - Ces fonctions réalisent une comparaison arithmétique entre deux opérandes, qui doivent être entiers. Elles retournent un entier qui est le plus grand, ou le plus petit, des deux opérandes.

@MOD - Cette fonction accepte deux opérandes de type entier et retourne la valeur entière de opérande1 modulo opérande2. Le résultat est indéfini si l'un ou l'autre des opérandes est négatif, ou si opérande2 est nul.

<, >, =, # - Ce sont des opérateurs relationnels valables sur des entiers. L'opérateur '#' teste l'inégalité. Ils retournent VRAI ou FAUX.

@AND, @OR, @XOR - Ce sont des opérations logiques s'effectuant sur les représentations sous forme de chaînes de bits des deux opérandes ou sur les valeurs logiques VRAI ou FAUX. Si les opérandes sont des chaînes de bits, elles retournent une chaîne de bits, si ce sont des valeurs logiques, alors elles retournent VRAI ou FAUX.

6.3 *Monadic operators and functions*

polish-expr → operand "," monad-f

monad-f → "@ABS" | "@NEG" | "@NOT" | "@ISDEF"

@ABS - The type of 'operand' must be integer. This function returns an integer which is the absolute value of 'operand.'

@NEG - The type of 'operand' must be integer. This function returns an integer which is 'operand' arithmetically negated.

@NOT - If the type of 'operand' is logical, the value returned is its logical complement. If the type of 'operand' is integer, the value returned is the one's complement of its value.

@ISDEF - This function returns the logical value TRUE if 'operand' contains no unassigned variables, and returns FALSE otherwise.

6.4 *Dyadic operators and functions*

polish-expr → operand1 "," operand2 "," dyad-f

dyad-f → "+" | "-" | "/" | "*" | "@MAX" | "@MIN" | "@MOD" |
"<" | ">" | "=" | "#" | "@AND" | "@OR" | "@XOR"

For noncommutative functions, the conventional operand ordering is used. For example, in the case of division, the result is the value of operand1 divided by the value of operand2.

+, -, /, * - These perform the obvious arithmetic operations on two operands, which must be integer. The result is integer. Divide truncates toward zero, and its result is undefined if operand2 is zero.

@MAX, @MIN - These functions perform an arithmetic comparison between two operands, which must be integer. They return an integer which is the greater, or lesser, of the two operands.

@MOD - This takes two integer operands and returns the integer value operand1 modulo operand2. The result is undefined if either operand is negative, or if operand2 is zero.

<, >, =, # - These are relational operators for integer values. The operator '#' tests inequality. They return TRUE or FALSE.

@AND, @OR, @XOR - These are logical operations on the bit string representations of the values of the two operands or on the logical values TRUE or FALSE. If the operands are bit strings, a bit string is returned; if they are logical values, then TRUE or FALSE is returned.

6.5 *Opérations de manipulations de données*

expression-polonaise → opérande1 "," opérande2 "," opérande3 "," "@EXT"

@EXT - Cette fonction extrait une chaîne de bits. Les opérandes doivent être des entiers non négatifs. Opérande1 est la valeur d'où est extraite la chaîne de bits.

Opérande2 et opérande3 sont les valeurs entières représentant la position de début et de fin de la chaîne de bits par rapport à l'extrémité droite de opérande1. La valeur est retournée avec justification à droite et remplissage par des zéros. Le bit de poids faible est numéroté zéro.

expression-polonaise → opérande1 "," opérande2 "," opérande3 "," opérande4 "," "@INS"

@INS - Cette fonction insère une chaîne de bits. Les opérandes doivent être des entiers non négatifs. La valeur de opérande2 est insérée à l'intérieur de opérande1. Opérande3 et opérande4 sont les valeurs entières représentant la position de début et de fin de la chaîne de bits par rapport à l'extrémité droite de la valeur de opérande1. La valeur de opérande2 est traitée comme une chaîne de bits et remplace la valeur de opérande1 entre les positions de début et de fin bornes incluses. La valeur de la fonction est la chaîne de bits compactée résultante. La troncature ou le remplissage par des zéros se fait par la gauche. Le bit de poids faible est numéroté zéro.

6.6 *Autres opérations*

expression-polonaise → valeur "," condition "," numerreur "," "@ERR"

valeur → expression

condition → expression

numerreur → expression

@ERR - Si condition est VRAI, alors un message d'erreur est généré, le numéro de l'erreur étant fourni par numerreur. Cette fonction retourne 'valeur.'

Par exemple, pour émettre #3 externe sous forme d'un octet et générer l'erreur #25 sur débordement:

LR(X3,X3,FF,>,25,@ERR,1).

expression-conditionnelle → condition "," "@IF" "," expression1 "," "@ELSE" "," expression2 "," "@END"

condition → expression

6.5 *Data manipulation operations*

polish-expr → operand1 "," operand2 "," operand3 "," "@EXT"

@EXT - This function extracts a bit string. The operands must be non-negative integers. Operand1 is the value from which the bit string is to be extracted.

Operand2 and operand3 are the integer values of the starting and ending bit positions relative to the right end of operand1. The value is returned right-justified with binary zero fill. The least significant bit is numbered zero.

polish-expr → operand1 "," operand2 "," operand3 "," operand4 "," "@INS"

@INS - This function inserts a bit string. The operands must be non-negative integers. The value of operand2 is inserted inside operand1. Operand3 and operand4 are the integer values of starting and ending bit positions relative to the right end of the value of operand1. The value of operand2 is treated as a bit string and replaces the value of operand1 between these starting and ending bit positions, inclusive. The value of the function is the resultant packed bit string. Truncation or binary zero fill is on the left. The least significant bit is numbered zero.

6.6 *Other operations*

polish-expr → value "," condition "," errornum "," "@ERR"

value → expression

condition → expression

errornum → expression

@ERR - If condition is TRUE, then an error message is generated, with errornum becoming the error number. This function returns 'value.'

For example, to emit external #3 into one byte and generate error #25 on overflow:

LR(X3,X3,FF,>,25,@ERR,1).

conditional-expr → condition "," "@IF" "," expression1 "," "@ELSE" "," expression2 "," "@END"

condition → expression

@IF - Si la valeur de 'condition' est VRAI, alors expression1 est évaluée et sa valeur et son type deviennent la valeur et le type de la fonction; expression2 n'est pas évaluée. Si la valeur de 'condition' est FAUX, alors expression1 n'est pas évaluée; expression2 est évaluée et sa valeur et son type deviennent la valeur et le type de la fonction.

7. Variables

Certaines variables internes des processus FUMOM (par exemple le pointeur de chargement ou les éléments du dictionnaire de symboles) doivent être manipulés selon les opérations exprimées dans le module objet. De telles variables sont appelées variables FUMOM et leur nom symbolique peut apparaître dans des expressions.

Les variables internes de FUMOM sont identifiées par un nom ayant la syntaxe suivante:

variable-FUMOM → lettre-non-hexadécimale nombre-hexadécimal?

La lettre-non-hexadécimale donne la classe de la variable. Si plusieurs variables de la même classe existent, un nombre-hexadécimal est nécessaire. Si ce nombre-hexadécimal fait référence au numéro de la section courante, il peut être omis.

Sauf pour la variable-X, les variables peuvent apparaître dans la partie gauche d'une commande-AS (voir section 12).

7.1 *Variable-G (adresse de début de l'exécution)*

variable-G → "G"

La variable-G contient l'adresse de la première instruction à exécuter. Cette valeur doit être chargée dans le compteur ordinal pour que le programme s'exécute. Le réalisateur doit définir les conséquences d'une absence d'affectation à la variable-G, ou les conséquences de multiples affectations à cette variable.

7.2 *Variables-I (définition externe-valeurs symboliques)*

variable-I → "I" nombre-hexadécimal

La variable-I est affectée d'une valeur qui doit être disponible pour les autres modules en vue de l'édition de liens. Cette valeur est aussi disponible dans le module où elle est définie par référence à la variable-I. La valeur est associée avec la définition externe spécifiée dans la commande-NI correspondante (voir 11.1). L'effet d'affectations multiples à une variable-I est indéfini.

7.3 *Variables-L (limite inférieure)*

variable-L → "L" numéro-section?

numéro-section → nombre-hexadécimal

@IF - If the value of 'condition' is TRUE, then expression1 is evaluated and its value and type become the value and type of the function; expression2 is not evaluated. If the value of 'condition' is FALSE, then expression1 is not evaluated; expression2 is evaluated and its value and type become the value and type of the function.

7. Variables

Some internal variables of MUFOM processes (for example load pointer or symbol dictionary items) shall be manipulated according to operations expressed in the object module. Such variables are called MUFOM variables and their symbolic names may appear in expressions.

The internal variables of MUFOM are identified by a name with the following syntax:

MUFOM-variable → nonhexletter hexnumber?

The nonhexletter gives the class of the variable. If several variables of the same class exist, a hexnumber is required. If this hexnumber refers to the current section number, it may be omitted.

Except for the X-variable, the variables can appear in the left part of an AS-command (see Section 12).

7.1 *G-variable (execution starting address)*

G-variable → "G"

The G-variable contains the address of the first instruction to be executed. This value is to be loaded into the program counter to run the program. The implementer shall define the results of not assigning to a G-variable, or assigning to it more than once.

7.2 *I-variables (external definition - symbol values)*

I-variable → "I" hexnumber

The I-variable is assigned a value which is to be made available to other modules for the purpose of linkage edition. That value is also available in the module in which it is defined by reference to the I-variable. The value is associated with the external definition specified in the corresponding NI-command (see 11.1). The effect of more than one assignment to an I-variable is undefined.

7.3 *L-variables (low limit)*

L-variable → "L" section-number?

section-number → hexnumber

Une section est chargée dans une zone d'un seul tenant de la mémoire de la machine cible. Les bornes d'une section sont données par son adresse de valeur inférieure (variable-L) et par sa taille (variable-S). La variable-L contient la valeur de l'adresse inférieure de la section spécifiée par le numéro-section. Il ne doit pas y avoir d'affectation à une variable-L dans une section FUMOM relogeable.

7.4 *Variables-N (noms)*

variable-N → "N" nombre-hexadécimal

Les variables-N sont utilisées pour la vérification de type dans la commande-TY (voir 11.5). Une variable-N fait référence au nom d'une variable utilisateur et est définie par la commande-NN (voir 11.3).

7.5 *Variables-P (pointeur de chargement)*

variable-P → "P" numéro-section?

numéro-section → nombre-hexadécimal

La variable-P de la section spécifiée par numéro-section contient l'adresse de la position mémoire de la machine cible où l'élément suivant de l'information binaire doit être chargé. La variable-P est incrémentée automatiquement lorsqu'un nouvel élément est chargé. Elle peut aussi être modifiée grâce à une commande-AS (voir section 11).

La valeur initiale de la variable-P est égale à la valeur de la variable-L (limite inférieure) de cette section, qui représente l'adresse de début de chargement.

7.6 *Variables-R (référence de relogement)*

variable-R → "R" numéro-section?

numéro-section → nombre-hexadécimal

La variable-R est un point de référence dans une section relogeable. Les adresses dans la section peuvent être spécifiées par rapport à la variable-R. La valeur initiale de la variable-R est égale à la valeur de la variable-L pour cette section.

Lorsque l'éditeur de liens combine plusieurs sections en une seule, il peut générer des affectations à la variable-R de la section résultat de manière à ce que les translations de relogement restent inchangées. Les adresses qui doivent être relogées utiliseront en général la variable-R de la section comme points de relogement.

7.7 *Variables-S (taille de section)*

variable-S → "S" numéro-section?

numéro-section → nombre-hexadécimal

Il y a une variable-S pour chaque section dans un module objet. La variable-S contient la taille (exprimée en UMA de la machine cible) de la section donnée par numéro-section. L'effet d'affectations multiples à une variable-S est indéfini. La variable-L et la variable-S d'une section absolue donnent les bornes de cette section. Les sections relogeables doivent être relogées en dehors de ces régions ainsi bornées.

A section is loaded within a contiguous region of the target memory. The bounds of a section are given by its lowest address (L-variable) and its size (S-variable). The L-variable contains the value of the low address for the section specified by the section-number. There shall not be an assignment to an L-variable in a relocatable MUFOM section.

7.4 *N-variables (name)*

N-variable → "N" hexnumber

The N-variables are used for type checking in the TY-command (see 11.5). An N-variable refers to the name of a user's variable and is defined by the NN-command (see 11.3).

7.5 *P-variables (load pointer)*

P-variable → "P" section-number?

section-number → hexnumber

The P-variable of the section specified by section-number contains the address of the target memory location where the next element of binary information is to be loaded. The P-variable is automatically incremented whenever a new element is loaded. It may also be modified using an AS-command (see Section 11).

The initial value of the P-variable is equal to the value of the L-variable (low limit) for that section, which is the starting load address.

7.6 *R-variables (relocation reference)*

R-variable → "R" section-number?

section-number → hexnumber

The R-variable is a reference point within a relocatable section. Addresses within the section may be specified relative to the R-variable. The initial value of the R-variable is equal to the value of the L-variable for that section.

When the linkage editor joins several sections into one, it may generate assignments to the R-variable of the resulting section in such a way that the relocation offsets remain unchanged. Addresses which are to be relocated will, in general, use as relocation points, the R-variable of the section.

7.7 *S-variables (section size)*

S-variable → "S" section-number?

section-number → hexnumber

There is one S-variable for each section in an object module. The S-variable contains the size (in target machine MAUs) of the section given by section-number. The effect of assigning to any S-variable more than once is undefined. The L-variable and S-variable of an absolute section give bounds to that section. Relocatable sections shall be relocated outside such bounded regions.

7.8 *Variables-W (registre de travail)*

variable-W → "W" nombre-hexadécimal

Aucune signification spéciale n'est attachée à ces variable - en différentes parties du même module, une variable-W peut être utilisée pour des buts différents (par exemple pour stocker temporairement une valeur).

7.9 *Variables-X (référence à un symbole externe)*

variable-X → "X" nombre-hexadécimal

Les variables-X correspondent aux références externes. Elles sont reliées aux variables-I des autres modules. Lorsqu'une commande-NX est rencontrée, elle est comparée avec les définitions externes données par les commandes-NI (voir 11.1). Si une définition externe est trouvée, la référence externe est résolue en donnant à la variable-X la valeur de la variable-I correspondante.

Par définition, les variables-X ne peuvent être placées dans la partie gauche d'une commande-AS. Elles apparaissent principalement en tant que paramètre pour les opérations de relogement. Le réalisateur doit spécifier l'action du chargeur lorsque celui-ci rencontre une variable-X.

8. Commandes au niveau module

8.1 *Commande MB (début de module)*

commande-MB → "MB" configuration-machine-cible
("," nom-module)? "."

configuration-machine-cible → identificateur

nom-module → chaîne-caractère

par exemple MB18080,07MONITOR.

La commande-MB doit être la première commande d'un module objet. La configuration-machine-cible peut définir le nom de la machine-cible, le type d'unité centrale et le système d'exploitation. Le nom-module devrait apparaître dans la carte d'implantation mémoire.

8.2 *Commande ME (fin de module)*

commande-ME → "ME".

La commande-ME doit être la dernière commande dans un module objet. Elle définit la fin du module.

8.3 *Commande DT (date et heure de création)*

commande-DT → "DT" digit* "."

7.8 *W-variables (working register)*

W-variable → "W" hexnumber

No special meaning is attached to these variables - in different parts of the same module, a W-variable may be used for different purposes (for example to store a value temporarily).

7.9 *X-variables (external symbol reference)*

X-variable → "X" hexnumber

The X variables correspond to external references. They are mapped onto the I-variables of other modules. When an NX command is encountered, it is compared with external definitions given by NI-commands (see 11.1). If an external definition is found, the external reference is resolved by setting the value of the X-variable to the value of the corresponding I-variable.

By definition, X-variables cannot be the left part of an AS-command. They occur mainly as parameters for relocation operations. The implementer shall specify the action of the loader if it should encounter an X-variable.

8. Module-level commands

8.1 *MB (module begin) command*

MB-command → "MB" target-machine-configuration
("," module-name)? "."

target-machine-configuration → identifier

module-name → char-string

for example MB18080,07MONITOR.

The MB-command shall be the first command of an object module. The target-machine-configuration may define the target-machine's name, CPU type, and operating system. The module-name should appear in the load map.

8.2 *ME (module end) command*

ME-command → "ME".

The ME-command shall be the last command in an object module. It defines the end of the module.

8.3 *DT (date and time of creation) command*

DT-command → "DT" digit* "."

Pour spécifier la date et l'heure de création de la sortie d'un processus, l'enregistrement de la date et de l'heure peut être utilisé. La première partie est l'année (4 chiffres), suivie du mois, du jour, de l'heure, de la minute, de la seconde (chacun avec 2 chiffres) et la fraction de seconde (avec un nombre de chiffres quelconque). Seules les parties les plus importantes de la date et de l'heure sont nécessaires; par exemple sur une machine qui possède seulement la date:

DT19810723

Cette forme est une approximation des normes ANSI X3.43 [2] et ISO 3307 [3]. L'absence de point avant la fraction de seconde est rendue obligatoire par la convention FUMOM de terminaison des commandes par un point.

8.4 *Commande AD (descripteur d'adresse)*

commande-AD → "AD" bits-par-UMA
("," UMA-par-adresse
("," ordre)?)? "."

bits-par-UMA → nombre-hexadécimal

UMA-par-adresse → nombre-hexadécimal

ordre → "L" | "M"

par exemple AD8,2,L. CO,05 8080.
AD24,1. CO,05 7094.

(Voir 9.1 pour une description de la commande "CO".)

De manière à faciliter l'utilisation de processus indépendants de la machine, la commande-AD est utilisée pour spécifier les formats d'adresse, la taille maximale d'une adresse et le nombre de bits dans une unité minimale adressable (UMA). La commande-AD a priorité sur les valeurs par défaut définies par le réalisateur.

Bits-par-UMA spécifie le nombre de bits dans une UMA.

UMA-par-adresse spécifie le nombre maximal d'UMA dans une adresse devant être relogées. Si l'UMA-par-adresse est omise, elle est prise par défaut égal à un. Un relogeur standard ou un éditeur de liens n'est pas obligé d'accepter un module pour lequel le produit de bits-par-UMA et UMA-par-adresse dans la commande-AD dépasse 64 (en décimales).

Le champ ordre de la commande-AD n'a besoin d'être spécifié que lorsque la taille maximale de l'adresse dépasse une UMA et que l'ordre est différent de la valeur par défaut M.

M spécifie que l'UMA la plus significative occupe l'adresse la plus basse dans la machine cible; L spécifie que l'UMA la moins significative occupe l'adresse la plus basse de la machine cible.

Un code d'ordre défini par le réalisateur peut spécifier un autre ordre possible. La commande-AD n'a pas d'effet sur l'éditeur de liens.

To specify the date and time of creation of the output of a process, the date and time record may be used. The first part is the year (4 digits), followed by month, day, hour, minute, second (each 2 digits), and the fraction of a second (any number of digits). Only the more significant portions of the date and time need be given; for example on a machine that has only a date:

DT19810723.

This form approximates the standards ANSI X3.43 [2] and ISO 3307 [3]. The absence of a period before the fraction of seconds is forced by the MUFOM convention of terminating a command by a period.

8.4 *AD (address descriptor) command*

AD-command → "AD" bits-per-MAU
("," MAUs-per-address
("," order)?)? "."

bits-per-MAU → hexnumber

MAUs-per-address → hexnumber

order → "L" | "M"

for example AD8,2,L. CO,05 8080.
AD24,1. CO,05 7094.

(See 9.1 for a description of the "CO" command.)

In order to facilitate the use of machine-independent processes, the AD-command is used to specify address formats, the maximum size of an address, and the number of bits in a minimum addressable unit (MAU). The AD-command overrides implementer-defined default values.

Bits-per-MAU specifies the number of bits in a MAU.

MAUs-per-address specifies the maximum number of MAUs in an address to be relocated. If the MAUs-per-address is omitted, it is assumed to be one. A standard relocator or loader need not accept any module for which the product of bits-per-MAU and MAUs-per-address in the AD-command exceeds (decimal) 64.

The order field of the AD-command need only be specified when the maximum address size exceeds one MAU and the order is other than the default value M.

M specifies that the most significant MAU occupies the lowest address in the target machine; L specifies that the least significant MAU occupies the lowest address in the target machine.

An implementer-defined order code may specify another possible ordering. The AD-command has no effect on the linker.

9. Commentaires et somme de contrôle

9.1 *Commentaires*

FUMOM est un langage intermédiaire qui n'est en général pas vu par l'utilisateur. Mais même si les modules objets ne sont pas destinés à être lus ou modifiés directement, des commentaires peuvent être utiles dans certains cas.

Par exemple, un processus peut générer une carte d'implantation mémoire décrivant la structure générale d'un module (qui peut être complexe s'il résulte de multiples éditions de liens). Pour faciliter la compréhension, des commentaires FUMOM peuvent être copiés dans la carte d'implantation mémoire.

Des commentaires peuvent aussi être utilisés pour conserver le nom du fichier source auquel le module correspond, pour insérer un indicateur annonçant si la compilation a été correcte, etc.

commande-CO → "CO" niveau-commentaire? "," texte-commentaire "."

niveau-commentaire → nombre-hexadécimal

texte-commentaire → chaîne-caractère

Par exemple CO3,1A I/O CONTROLLER (15 JUNE 81).

Le texte-commentaire dans une commande-CO peut-être utilisé par un processus pour transmettre de l'information à l'utilisateur. Le niveau-commentaire peut être utilisé par le processus pour sélectionner ou exclure le commentaire de la transmission ou de la sortie. L'interprétation du niveau-commentaire doit être définie par le réalisateur. Un processus peut ne pas prendre en compte une commande-CO, ou encore peut copier cette commande en modifiant seulement le niveau-commentaire, si on le désire.

9.2 *Commande CS (somme de contrôle)*

commande-CS → "CS" somme-de-contrôle? "."

somme-de-contrôle → digit-hexadécimal digit-hexadécimal

La somme de contrôle est formée par l'addition non signée des valeurs binaires des caractères ASCII successifs (à l'exception des caractères de contrôle) dont on fait le total (en opérant modulo 128). Le dernier octet à être additionné est le "S" de la commande-CS. La somme de contrôle devrait correspondre avec (être égale à) la valeur entière de digit-hexadécimal digit-hexadécimal ci-dessus. La somme de contrôle courante est remise à zéro à la fois au début d'un module et après la rencontre du point terminant une commande-CS. Un simple "CS" remet la somme de contrôle courante à zéro sans effectuer la vérification. Les commandes de somme de contrôle peuvent être insérées n'importe où les autres commandes peuvent être placées.

9. Comments and checksum

9.1 *Comments*

MUFOM is an intermediate language which is usually not seen by the user. But even if object modules are not intended to be read or directly modified, comments may be useful in some cases.

For example, a process may generate a load map describing the overall structure of a module (which can be complex if it is the result of multiple linkage editions). To facilitate comprehension, MUFOM comments may be copied into the load map.

Comments may also be used to record the name of the source file to which the module corresponds, to insert a flag indicating if the compilation was successful, etc.

CO-command → "CO" comment-level? "," comment-text "."

comment-level → hexnumber

comment-text → char-string

For example CO3,1AI/O CONTROLLER (15 JUNE 81).

The comment-text in a CO-command may be used by a process to pass information to the user. The comment-level may be used by the process to select or exclude the comment from transmission or output. The interpretation of the comment-level shall be implementer-defined. A process may discard a CO-command or it may copy the CO-command, changing only the comment-level, if desired.

9.2 *CS (checksum) command*

CS-command → "CS" checksum? "."

checksum → hexdigit hexdigit

The checksum is formed by the unsigned binary adding of the binary values of successive ASCII characters (control characters are not added) together (modulo 128). The last byte to be added is the "S" of the CS-command. The checksum should agree with (be equal to) the integer value of hexdigit hexdigit above. The running checksum is reset to zero both at the start of a module and after encountering the period terminating a CS-command. A simple "CS." resets the running checksum to zero without checking it. Checksum commands may be inserted anywhere any other command might be expected.

10. Sectionnement

Un module objet peut être divisé en une ou plusieurs sections. Une section est une région du programme gérée séparément. Elle possède un type qui influence globalement les actions effectuées selon les commande qu'elle contient.

A l'intérieur d'un même module, une section est identifiée par un "numéro de section". Une section est introduite (ou est reprise) par une commande-SB contenant son numéro, et elle se termine (ou est suspendue) par une nouvelle commande-SB avec un numéro de section différent ou par une commande-ME.

10.1 *Commande SB (début de section)*

commande-SB → "SB" numéro-section "."

numéro-section → nombre-hexadécimal

par exemple SB5.

Le réalisateur doit spécifier la valeur maximale de numéro-section. Toutes les valeurs de numéro-section entre zéro et la valeur maximale (incluse) doivent être valables.

Jusqu'à ce qu'une commande-SB soit rencontrée, le numéro de section est zéro.

10.2 *Commande ST (type de section)*

commande-ST → "ST" numéro-section ("," type-section)*
("," nom-section)? "."

numéro-section → nombre-hexadécimal

type-section → lettre

nom-section → chaîne-caractère

Exemples de types de section:

ST0,R,U,04SUBR.	CO,0D section de code.
ST1,S.	CO,0D donnée non nommée.
ST23,E,03COM.	CO,0D commun nommé.
ST1F,M.	CO,0D commun blanc.
ST4,Z,S.	CO,12 donnée en page zéro.

Dans une commande-ST, type-section définit le type de la section identifiée par le numéro-section. S'il est présent, nom-section donne un nom symbolique à cette section.

Une section peut être absolue ou relogeable. Une section est absolue lorsque ses adresses de chargement sont absolues (connues lors de l'assemblage ou de la compilation). Une section est relogeable lorsque ses adresses de chargement sont relatives à une base de translation dont la valeur n'est pas connue au départ.

10. Sectioning

An object module can be divided into one or more sections. A section is a separately controlled region of the program. It has a type, which globally influences the actions performed in accordance with the commands that it contains.

Within the same module, a section is identified by a "section-number". A section is introduced (or resumed) by an SB-command containing its number, and it is terminated (or suspended) by a new SB-command with a different section-number or by an ME-command.

10.1 *SB (section begin) command*

SB-command → "SB" section-number "."

section-number → hexnumber

for example SB5.

The implementer shall specify the maximum possible section-number. All section-numbers between zero and the maximum (inclusive) shall be valid.

Until an SB-command is encountered, the section-number is zero.

10.2 *ST (section type) command*

ST-command → "ST" section-number ("," section-type)*
("," section-name)? "."

section-number → hexnumber

section-type → letter

section-name → char-string

Examples of section types:

ST0,R,U,04SUBR.	CO,0D code section.
ST1,S.	CO,0D unnamed data.
ST23,E,03COM.	CO,0D named common.
ST1F,M.	CO,0D blank common.
ST4,Z,S.	CO,12 data in zero page.

In an ST-command, section-type defines the type of the section identified by section-number. If present, section-name gives a symbolic name to that section.

A section can be absolute or relocatable. A section is absolute when its loading addresses are absolute (known at assembly or compile time). A section is relocatable when its loading addresses are relative to a relocation base whose value is initially not known.

Les sections relogeables peuvent être chargées individuellement dans des zones mémoire séparées. Cependant, dans certains cas, les adresses de chargement des sections doivent être liées. Lors du processus de chargement, l'utilisateur peut avoir le contrôle complet de l'implantation en mémoire si le chargeur le permet: par exemple en utilisant des directives de segmentation.

Excepté dans les cas spéciaux (par exemple la segmentation), FUMOM permet l'interdépendance entre les sections de différents modules. Ainsi, lors de l'édition de liens, la structure des modules est construite de manière à correspondre à la structure des programmes sources.

Pour l'éditeur de liens, il est possible d'exprimer trois sortes de relations mutuellement exclusives entre les sections.

1) Plusieurs sections doivent être combinées en une seule.
2) Plusieurs sections doivent se chevaucher.
3) Les sections ne doivent pas coexister.

Comme ces sections apparaissent habituellement dans des modules objets différents, la relation est établie selon leurs types et leurs noms symboliques.

FUMOM classe les sections de quatre manières. Elles fournissent au relogeur les informations nécessaires pour implanter les sections de manière appropriée. Les codes ou les catégories peuvent être étendues pour les machines cibles où la norme n'est pas suffisante. Il peut être significatif de spécifier plus, ou moins, d'un attribut de chaque catégorie. Quelques combinaisons d'attributs ne sont pas significatives.

Accès: Cet attribut contrôle la manière dont le relogeur groupe les sections ensemble et les implante en mémoire.

Inscriptible (RAM, code W). C'est l'option par défaut dans une commande-ST si aucun attribut d'accès n'est spécifié.

Lecture seule (ROM, code R)

Exécution seule (code X)

Page zéro (code Z)

Abs (code A). Il doit exister une affectation à la variable-L.

Nommé: Cette propriété est dérivée de la présence d'un nom.

Chevauchement: Cet attribut contrôle ce qu'il convient de faire avec deux sections si elles possèdent le même attribut de nom et d'accès. (Deux sections non nommées sont dites posséder le même nom si elles ont le même attribut d'accès.)

Egal (code E). Erreur si les longueurs diffèrent.

Max (code M). Utilise la longueur la plus grande qui a été rencontrée (par exemple un commun blanc).

Relocatable sections may be individually loaded into separate storage areas. However, in some cases, the loading addresses of sections shall be related. During the loading process, the user may have full control of the storage layout if the loader allows: for example by using overlay directives.

Except in special cases (for example overlays), MUFOM allows inter-dependency between sections of different modules. Thus, during the linkage edition, the module structure is built to correspond to the structure of the source programs.

For the linkage editor, it is possible to express three mutually exclusive kinds of relations among sections:

1) Several sections are to be joined into a single one.
2) Several sections are to be overlapped.
3) Sections are not to coexist.

As these sections usually appear in different object modules, the relationship is established according to their types and symbolic names.

MUFOM categorizes sections in four ways. These provide the relocator with the information needed to lay out sections appropriately. The codes or categories may be extended for target machines where the standard is not sufficient. It may be meaningful to specify more, or less, than one attribute from each category. Some combinations of attributes are not meaningful.

Access: This attribute controls how the relocator groups sections together and lays them out in memory.

Writable (RAM, code W). This is the default in an ST-command if no access attribute is specified.

Read only (ROM, code R)

Execute only (code X)

Zero page (code Z)

Abs (code A). There shall be an assignment to the L-variable.

Named: This property is derived from the presence of the name.

Overlap: This attribute controls what to do with two sections if they have the same name and access attribute. (Two unnamed sections are said to have the same name if they have the same access attribute.)

Equal (code E). Error if lengths differ.

Max (code M). Use largest length encountered (for example blank common).

Unique (code U). Les noms devraient être uniques (par exemple code).

Cumulatif (code C). Effectue la concaténation des sections. La section résultante doit être alignée de manière à préserver les alignements de ses composants (voir 10.3).

Séparé (code S). Pas de connexion entre les sections. Des sections multiples peuvent avoir le même nom et le relogeur peut leur allouer des emplacements indépendants en mémoire.

Quand allouer:

Maintenant (code N). C'est le cas normal. C'est l'option par défaut si aucune n'est spécifiée.

Remettre (code P). Le relogeur doit effectuer les allocations après toutes les sections dotées de l'attribut "maintenant", fournissant ainsi un moyen d'obtenir "la dernière adresse allouée".

Dans un module objet, si la commande-ST est présente, elle peut précéder la commande-SB correspondante. Si aucune commande-ST n'est donnée pour une section, son type est absolu (A).

10.3 *Commande SA (alignement de section)*

commande-SA → "SA" numéro-section "," frontière-UMA? ("," taille-page)? "."

frontière-UMA → expression

taille-page → expression

EXEMPLES:

Début de la section 1 sur une frontière de 2-UMA:

SA1,2.

Contrainte de la section 2 à résider dans une page de 256 UMA:

SA2,,100.

Lorsqu'une section doit débuter sur une frontière qui est un multiple entier de plus d'une UMA, on utilise la commande-SA.

Si la taille de la page est indiquée, la section peut ne pas être implantée avec chevauchement d'une frontière de page.

11. Nom symbolique et déclaration de type

11.1 *Commande NI (nom d'un symbole interne)*

commande-NI → "N" variable-I "," nom-def-ext "."

variable-I → "I" nombre-hexadécimal

Unique (code U). Names should be unique (for example code).

Cumulative (code C). Concatenate sections together. The resulting section shall be aligned in such a way as to preserve the alignments of its components (see 10.3).

Separate (code S). No connection between sections. Multiple sections can have the same name, and the relocator may allocate them at unrelated places in memory.

When to allocate:

Now (code N). This is the normal case. This is the default if none is specified.

Postpone (code P). Relocator must allocate after all "now" sections, thereby providing a way of getting "the last address allocated".

In an object module, if the ST-command is present, it may precede the corresponding SB-command. If no ST-command is given for a section, its type is absolute (A).

10.3 *SA (section alignment) command*

SA-command → "SA" section-number "," MAU-boundary? ("," page-size)? "."

MAU-boundary → expression

page-size → expression

EXAMPLES:

Start section 1 on a 2-MAU boundary:

SA1,2.

Force section 2 to reside in a 256-MAU page:

SA2,,100.

When a section is required to start on a boundary which is some integral multiple of more than one MAU, the SA-command is used.

If page size is supplied, the section may not be allocated across a page boundary.

11. Symbolic name and type declaration

11.1 *NI (name of internal symbol) command*

NI-command → "N" I-variable "," ext-def-name "."

I-variable → "I" hexnumber

nom-def-ext → chaîne-caractère

par exemple NIA,04SINE.

Une commande NI doit être fournie pour chaque symbole qui est défini dans le module courant et qui peut être référencé à partir d'un autre module. Cela est appelé définition externe. Elle indique que dans le module courant la variable-I (voir 7.2) est associée avec la définition externe de nom-def-ext. L'effet de plus d'une commande-NI lors de la même édition de liens avec le même nom-def-ext est indéfini. Une édition de commande-NI doit toujours apparaître avant toutes les occurrences de sa variable-I correspondante.

Quelques noms symboliques qui apparaissent dans les programmes sources doivent être conservés dans les modules objets en vue d'un traitement ultérieur (par exemple l'édition de liens). Dans FUMOM, chaque symbole est déclaré par une commande séparée. De telles commandes débutent avec le caractère "N" suivi d'une lettre qui détermine la classe de ce nom. Ces commandes sont employées pour la résolution des références externes et pour la spécification des attributs associés aux noms.

Si le module produit par l'éditeur de liens peut être à nouveau relié dans une étape ultérieure, les commandes-NI sont conservées à moins que le réalisateur ne fournisse une option à l'éditeur de liens en vue de les supprimer (voir section 14).

11.2 *Commande-NX (nom d'un symbole externe)*

commande-NX → "N" variable-X "," nom-ref-ext "."

variable-X → "X" nombre-hexadécimal

nom-ref-ext → chaîne-caractère

par exemple NX11,04SINE

Une commande-NX doit être fournie pour chaque symbole externe qui est référencé dans le module courant. Elle indique que dans le module courant la variable-X (voir 7.9) est associée avec la référence externe de nom-ref-ext.

Une commande-NX doit toujours précéder toute occurrence de sa variable-X correspondante.

Puisque le module produit par l'éditeur de liens peut être à nouveau relié dans une étape ultérieure, les commandes-NX qui correspondent à des références non résolues sont conservées.

11.3 *Commande NN (nom)*

commande-NN → "N" variable-N "," nom "."

variable-N → "N" nombre-hexadécimal

nom → chaîne-caractère

ext-def-name → char-string

for example NIA,04SINE.

An NI command shall be provided for each symbol which is defined in the current module and which may be referenced from another module. This is called external definition. It indicates that in the current module the I-variable (see 7.2) is associated with the external definition of ext-def-name. The effect of more than one NI-command in the same linkage edition with the same ext-def-name is undefined. An NI-command edition shall always come before all occurrences of its corresponding I-variable.

Some symbolic names which appear in source programs must be retained in the object modules for further processing (for example linkage edition). In MUFOM, each symbol is declared by a separate command. Such commands begin with the character "N" followed by a letter which determines the class of that name. These commands are employed in the resolution of external references and in the specification of attributes associated with names.

If the module produced by the linkage editor may be linked in a further step, the NI-commands are retained unless the implementer gives an option to the linkage editor to suppress them (see Section 14).

11.2 *NX (name of external symbol) command*

NX-command → "N" X-variable "," ext-ref-name "."

X-variable → "X" hexnumber

ext-ref-name → char-string

for example NX11,04SINE.

An NX-command shall be provided for each external symbol which is referenced in the current module. It indicates that in the current module the X-variable (see 7.9) is associated with the external reference of ext-ref-name.

An NX-command shall always precede any occurrence of its corresponding X-variable.

Since the module produced by the linkage editor may be linked in a further step, the NX-commands which correspond to unresolved references are retained.

11.3 *NN (name) command*

NN-command → "N" N-variable "," name "."

N-variable → "N" hexnumber

name → char-string

Une commande-NN doit être fournie pour chaque symbole qui est défini dans le module courant et qui ne peut pas être référencé à partir d'un autre module (c'est-à-dire qu'il est local au module). Elle peut être transmise à un outil de mise au point symbolique, ou elle peut fournir des symboles à la table des types. Elle indique que dans le module courant la variable-N correspondante (voir 7.4) est associée avec la définition locale du nom. L'effet de plus d'une commande-NN dans le même module avec le même nom est indéfini. Une commande-NN doit toujours apparaître avant toutes les occurrences de sa variable-N correspondante.

11.4 *Commande AT (attributs)*

commande-AT → "AT" variable "," entrée-table-type
("," niveau-lex ("," nombre-hexadécimal)*)? "."

variable → variable-I | variable-N | variable-X

entrée-table-type → nombre-hexadécimal

niveau-lex → nombre-hexadécimal

La commande-AT est utilisée pour définir les attributs d'un symbole tel que le type ou l'information nécessaire à la mise au point. Les attributs de nom et la valeur (c'est-à-dire l'adresse de chargement) sont définis par d'autres commandes. L'entrée-table-type identifie une entrée dans la table des types qui définit le type de ce symbole. Le niveau-lex peut être utilisé pour montrer la portée dans les modules des langages à structure de blocs; les nombres hexadécimaux supplémentaires sont définis par le réalisateur. Normalement, la commande-AT désigne les variables I, N ou X. Dans le dernier cas la vérification de type est effectuée, et l'entrée-table-type pour chaque variable-X doit être compatible avec l'entrée-table-type pour la variable-I correspondante (voir commande-TY ci-après).

11.5 *Commande TY (type)*

commande-TY → "TY" entrée-table-type ("," paramètre)+ "."

entrée-table-type → nombre-hexadécimal

paramètre → nombre-hexadécimal | variable-N | "T" entrée-table-type

La commande-TY spécifie une entrée dans la table des types. L'entrée-table-type est un numéro d'accès qui est utilisé dans ce module pour faire référence à cette entrée. Le reste de la commande spécifie une liste de paramètres de type. Un paramètre nombre-hexadécimal est un littéral; une variable-N fait référence à la chaîne de caractères qui représente le nom de cette variable; et un nombre-T fait référence de manière récursive à une autre entrée dans la table des types.

La compatibilité des types est établie entre deux entrées si les chaînes représentent des arbres de types identiques: c'est-à-dire qu'ils possèdent le même nombre de paramètres ayant la même forme; les paramètres numériques sont respectivement égaux; les entrées de variables-N se réfèrent à des variables ayant les mêmes noms; et les références de l'arbre des types concernent des entrées de table des types qui sont compatibles.

An NN-command shall be provided for each symbol which is defined in the current module and which may not be referenced from any other module (that is, it is local to the module). It may be passed to a symbolic debugger, or it may provide symbols for the type table. It indicates that in the current module the corresponding N-variable (see 7.4) is associated with the local definition of name. The effect of more than one NN-command in the same module with the same name is undefined. An NN-command shall always come before all occurrences of its corresponding N-variable.

11.4 *AT (attributes) command*

AT-command → "AT" variable "," type-table-entry
("," lex-level ("," hexnumber)*)? "."

variable → I-variable | N-variable | X-variable

type-table-entry → hexnumber

lex-level → hexnumber

The AT-command is used to define attributes of a symbol such as type or debugging information. The name and value (that is, load address) attributes are defined by other commands. The type-table-entry identifies an entry in the type table which defines the type for this symbol. The lex-level may be used to show the scope in back-structured language modules; additional hexnumbers are implementer-defined. Normally the AT-command designates I-, N-, or X-variables. In the last case, type checking is performed, and the type-table-entry for each X-variable must be compatible with the type-table-entry for the corresponding I-variable (see the TY-command below).

11.5 *TY (type) command*

TY-command → "TY" type-table-entry ("," parameter)+ "."

type-table-entry → hexnumber

parameter → hexnumber | N-variable | "T" type-table-entry

The TY-command specifies an entry in the type table. The type-table-entry is an access number that is used in this module to refer to this entry. The remainder of the command specifies a list of type parameters. A hexnumber parameter is a literal; an N-variable refers to the character string that is the name of that variable; and a T-number refers recursively to another entry in the type table.

Type compatibility is established between two entries if the strings represent identical type trees: that is, they have the same number of parameters of the same form; the numeric parameters are respectively equal; the N-variable entries refer to variables with the same names; and the type tree references refer to compatible type table entries.

Des types "joker" sont définis avec des entrées égales à zéro: un paramètre numérique de zéro est considéré comme pouvant être égal à n'importe quel paramètre numérique; N0 est considéré comme un nom équivalent à celui de tout autre variable-N; et T0 est considéré comme étant compatible avec n'importe quel type.

EXEMPLE: dans la table suivante, les types T3 et T4 sont compatibles:

T1,13,N0,88.

T2,13,N8,0.

T3,44,T7,T1.

T4,44,T0,T2.

12. Commande AS (affectation)

commande-AS → "AS" variable-FUMOM "," expression "."

par exemple ASP,A00.
AS17,R,8C,+.

La valeur de expression est affectée à la variable.

13. Commandes de chargement

La partie la plus importante d'un module objet est formée par la représentation interne des instructions, des adresses et des données telles qu'elles sont traduites à partir du programme source. A cause de l'utilisation séquentielle des adresses mémoire, il est possible d'isoler des *blocs* là où l'information est destinée à être chargée dans des emplacements contigus. Si toute l'information est complètement connue, comme dans le cas du chargement absolu, chaque bloc représente une image exacte (codée sous forme d'une représentation FUMOM) d'une région mémoire après le chargement. Dans ce cas, il est possible de spécifier le bloc par une ou plusieurs commandes-LD successives. Dans le cas contraire, l'information peut être divisée entre des commandes LD et LR qui sont successivement chargées dans l'ordre de leur traitement.

13.1 *Commande LD (chargement)*

commande-LD → "LD" const-chargement + "."

const-chargement → digit-hexadécimal +

par exemple LD1D7F1D8033C63006A144A144A16010415A
2B2A3D2D403C3E.

La const-chargement est une valeur qui est chargée dans une ou plusieurs UMA. La const-chargement doit être un nombre fixe de chiffres avec le plus significatif en tête. Le nombre de chiffres doit être défini par le réalisateur. Les seuls processus qui sont spécifiques d'une machine cible particulière (par exemple les générateurs de code, les chargeurs) sont nécessaires pour coder ou décoder les constantes de chargement; l'éditeur de liens et le relogeur n'utilisent ni ne modifient les commandes-LD, mais ne font que les copier.

"Don't care" types are defined with entries of zero: a numeric parameter of zero is considered equal to any numeric parameter; N0 is considered to be the same name as any other N-variable; and T0 is considered to be compatible with any type.

EXAMPLE: in the following table, types T3 and T4 are compatible:

T1,13,N0,88.

T2,13,N8,0.

T3,44,T7,T1.

T4,44,T0,T2.

12. AS (assignment) command

AS-command → "AS" MUFOM-variable "," expression "."

for example ASP,A00.
AS17,R,8C,+.

The value of expression is assigned to the variable.

13. Loading commands

The most important part of an object module is formed by the internal representation of instructions, addresses, and data as translated from the source program. Due to the sequential use of memory addresses, it is possible to isolate *blocks* where the information is destined to be loaded into contiguous locations. If all the information is completely known, as is the case for absolute loading, each block represents an exact image (encoded in the MUFOM representation) of a memory region after the loading. In such a case, the block may be specified by one or more successive LD-commands. Otherwise the information may be divided between LD and LR commands, which are successively loaded in the order processed.

13.1 *LD (load) command*

LD-command → "LD" load-constant + "."

load-constant → hexdigit +

for example LD1D7F1D8033C63006A144A144A16010415A
2B2A3D2D403C3E.

The load-constant is a value which is loaded into one or more MAUs. The load-constant shall be a fixed number of digits with the most significant first. The number of digits shall be defined by the implementer. Only processes which are specific to a particular target machine (for example code generators, loaders) are required to encode or decode load constants; the linker and relocator do not use or modify the LD-commands, but only copy them.

13.2 *Commande IR (initialiser la base de translation)*

commande-IR → "IR" lettre-translation ","
base-translation ("," nombre-de-bits)? "."

lettre-translation → lettre-non-hexadécimale

base-translation → expression

nombre-de-bits → expression

par exemple IRQ,X4,10.
IRG,R3,45,-,8.
IRT,XA,3.

La commande-IR initialise une base de translation (désignée par la lettre-translation) afin qu'elle puisse être utilisée dans la commande LR (charger-reloger). La base de translation est évaluée par le relogeur au moment où la commande IR est rencontrée et est utilisée pour décaler chaque adresse qui désigne cette base dans toutes les commandes-LR ultérieures, jusqu'à ce qu'une autre commande IR réinitialise cette base.

Le nombre-de-bits associe une taille de champ d'adressage avec cette base de translation pour toutes les références ultérieures. Cette taille ne doit pas excéder le produit de UMA-par-adresse et bits-par-UMA comme il est spécifié dans la commande-AD (voir 8.4). Si le nombre-de-bits n'est pas spécifié, il est pris égal à ce produit.

13.3 *Commande LR (charger reloger)*

commande-LR → "LR" charger-élément + "."

charger-élément → lettre-translation décalage "," | const-chargement |
"(" valeur-chargement ("," nombre-de-UMA)? ")"

lettre-translation → lettre-non-hexadécimale

décalage → nombre-hexadécimal

valeur-chargement → expression

nombre-de-UMA → expression

par exemple LR0000HB2,0001(P,10,+,2).

La commande-LR est une notation compacte pour l'information de chargement et de relogement. Trois sortes différentes de charger-élément peuvent coexister dans la commande.

Un charger-constante est du code absolu (non relogé) qui doit être chargé dans autant d'UMA qu'il est nécessaire, sous la même forme que la commande-LD (0000 et 0001 dans l'exemple ci-dessus).

13.2 *IR (initialize relocation base) command*

IR-command → "IR" relocation-letter ","
relocation-base ("," number-of-bits)? "."

relocation-letter → nonhexletter

relocation-base → expression

number-of-bits → expression

for example IRQ,X4,10.
IRG,R3,45,-,8.
IRT,XA,3.

The IR-command initializes a relocation-base (designated by the relocation-letter) for use in the LR (load-relocate) command. The relocation-base is evaluated by the relocator at the time the IR command is encountered and used to offset each address that designates this base in all subsequent LR-commands, until another IR-command reinitializes this base.

The number-of-bits associates an address field size with this relocation-base, for all subsequent references. This size shall not exceed the product of MAUs-per-address and bits-per-MAU as specified in the AD-command (see 8.4). If number-of-bits is not specified, it is equal to that product.

13.3 *LR (load relocate) command*

LR-command → "LR" load-item + "."

load-item → relocation-letter offset "," | load-constant |
"(" load-value ("," number-of-MAUs)? ")"

relocation-letter → nonhexletter

offset → hexnumber

load-value → expression

number-of-MAUs → expression

for example LR0000HB2,0001(P,10,+,2).

The LR-command is a compact notation for loading and for relocating information. Three different kinds of load-item may coexist in the command.

A load-constant is an absolute (unrelocated) code that is to be loaded into as many MAUs as are required, in the same form as the LD-command (0000 and 0001 in the example above).

Un décalage précédé par la lettre-translation (HB2 ci-dessus) est ajouté à la base de translation désignée, qui aura été spécifiée par sa commande-IR la plus récente (voir 13.2). Autant d'UMA sont chargées qu'il est nécessaire pour contenir le nombre de bits spécifié dans la commande-IR.

Une expression mise entre parenthèses permet des expressions de translation arbitraires (valeur-chargement). Le nombre d'UMA, s'il est omis, est présumé être égal à UMA-par-adresse spécifiée dans la commande-AD (voir 8.4).

Le nombre d'UMA spécifié ou pris par défaut est chargé avec la valeur de l'expression, justifiée à droite (c'est-à-dire que les bits les plus significatifs sont ignorés ou remplis par des zéros, selon les cas. Dans l'exemple ci-dessus, le dernier élément de chargement est seize plus le compteur ordinal P, remplissant 2 UMA.

Si, dans l'exemple ci-dessus, on suppose que P est initialement égal à 0123 hexadécimal et que la base-translation H a été initialisée avec une valeur hexadécimale de 1A et une largeur de champ de 5 bits, et qu'une UMA est définie comme étant de 16 bits, alors le résultat de cette commande LR est la configuration de bits suivante (en hexadécimal):

0000 000C 0001 0000 0136

La première UMA résulte de la constante de chargement 0000. La deuxième résulte de la lettre-translation H avec le décalage B2 (noter que dans la somme de 1A + B2 la retenue du cinquième bit est supprimée). La troisième UMA résulte de la constante de chargement 0001. Les quatrième et cinquième résultent de l'expression entre parenthèses (noter que P s'est incrémenté jusqu'à 0126 au point d'évaluation).

13.4 *Commande RE (copie)*

commande-RE → "RE" expression "."

par exemple RE A.LR0000.
RE64.LR(7F800000,P,+,4).

La commande-RE évalue l'expression pour déterminer le nombre de fois que la commande-LR qui la suit immédiatement doit être copiée (voir 13.3). La longueur maximale de la portion de la commande-LR à traiter par la commande-RE dépend de la réalisation, mais elle doit être d'au moins 64 caractères.

14. Edition de liens

Cette section contient les commandes qui se réfèrent seulement à l'édition de liens. Elles donnent des commandes supplémentaires sur les actions de l'éditeur de liens. Les actions de ces commandes peuvent de plus dépendre des options qui sont disponibles sur l'éditeur de liens. Par exemple, l'option de conserver les commandes-NI (voir 11.1) peut être annulée par une option de l'éditeur de liens à la disposition de l'utilisateur.

An offset preceded by relocation-letter (HB2 above) is added to the designated relocation base, which will have been specified by its most recent IR-command (see 13.2). As many MAUs are loaded as are required to contain the number of bits specified in the IR-command.

A parenthesized expression permits arbitrary relocation expressions (load-values). The number-of-MAUs, if omitted, is assumed to be the MAUs-per-address specified in the AD-command (see 8.4).

The specified or default number of MAUs is loaded with the value of the expression, right-justified (that is, the most significant bits are discarded or filled out with binary zeros, as appropriate). In the example above, the last load item is sixteen plus the location counter P, filling 2 MAUs.

If, in the example above, it is assumed that P is initially hex 0123 and relocation-base H has been initialized with a value of hex 1A and a field width of 5 bits, and a MAU is defined as 16 bits wide, then the result of that LR command is the following bit pattern (shown in hex):

0000 000C 0001 0000 0136

The first MAU results from the load constant 0000. The second MAU results from the relocation-letter H with offset B2 (note that the sum of 1A + B2 has the carry-out of the fifth bit suppressed). The third MAU results from the load constant 0001. The fourth and fifth MAUs result from the expression in parentheses (note that P has incremented to 0126 at the point of evaluation).

13.4 *RE (replicate) command*

RE-command → "RE" expression "."

for example RE4.LR0000.
RE64.LR(7F800000,P,+,4).

The RE-command evaluates the expression to determine the number of times the immediately following LR-command is to be replicated (see 13.3). The maximum length of the LR-command portion to be processed by the RE-command is implementation-dependent, but it shall be at least 64 characters.

14. Linkage edition

This section contains commands which refer only to linkage edition. They give additional control over the actions of the linkage editor. The actions of these commands may further depend on options available with the linkage editor. For example, the option to retain NI-commands (see 11.1) might be overriden by a user option given to the linkage editor.

14.1 *Commande RI (conserver le symbole interne)*

commande-RI → "R" variable-I ("," numéro-niveau)? "."

variable-I → "I" nombre-hexadécimal

numéro-niveau → nombre-hexadécimal

Cette commande indique que l'information symbolique définie auparavant par une commande-NI (avec la même variable-I) doit être conservée dans les modules objets produits par tout traitement ultérieur. Le numéro-niveau optionnel peut être utilisé pour spécifier dans quelles circonstances le symbole doit être conservé.

14.2 *Commande WX (symbole externe faible)*

commande-WX → "W" variable-X ("," valeur-par-défaut)? "."

variable-X → "X" nombre-hexadécimal

valeur-par-défaut → expression

Cette commande indique que le symbole externe spécifié par une commande-NX précédente (avec la même variable-X) doit être marquée comme externe faible.

Au cas où la référence externe n'est pas satisfaite, l'expression évaluée est la valeur de la variable-X.

15. Bibliothèques

Ces commandes se réfèrent de manière spécifique au problème du chargement de routines qui sont conservées dans des bibliothèques. Elles sont supposées fournir des options par défaut acceptables et surmonter le problème de deux ou plusieurs routines ayant le même nom pouvant être présenter dans des bibliothèques séparées.

15.1 *Commande LI (spécifier une liste de recherche par défaut en bibliothèque)*

commande-LI → "LI" chaîne-caractère ("," chaîne-caractère)* "."

Cette commande spécifie l'ordre par défaut selon lequel les bibliothèques doivent être examinées pour résoudre les références externes non satisfaites. Chaîne-caractère identifie une bibliothèque, et la recherche est effectuée dans les bibliothèques spécifiées par ces chaînes-caractères dans l'ordre de gauche à droite. Une commande LI annule toute commande-LI antérieure dans le même module.

15.2 *Commande LX (bibliothèque externe)*

commande-LX → "L" variable-X ("," chaîne-caractère) + "."

La commande-LX spécifie la ou les bibliothèques à partir desquelles la référence externe individuelle correspondant à une variable-X doit être satisfaite.

14.1 *RI (retain internal symbol) command*

RI-command → "R" I-variable ("," level-number)? "."

I-variable → "I" hexnumber

level-number → hexnumber

This command indicates that the symbolic information previously defined by an NI-command (with the same I-variable) is to be retained in the object modules produced by any subsequent processing. The optional level number may be used to specify under what circumstances the symbol is to be retained.

14.2 *WX (weak external symbol) command*

WX-command → "W" X-variable ("," default-value)? "."

X-variable → "X" hexnumber

default-value → expression

This command indicates that the external symbol specified by a previous NX-command (with the same X-variable) is to be flagged as a weak external.

In the case that the external reference is unsatisfied, the evaluated expression is the value of the X-variable.

15. Libraries

These commands refer specifically to the problem of loading routines which are contained in libraries. They are meant to give suitable defaults and to overcome the problem of two or more routines with the same name being present in separate libraries.

15.1 *LI (specify default library search list) command*

LI-command → "LI" char-string ("," char-string)* "."

This command specifies the default order in which libraries are to be searched to resolve unsatisfied external references. Char-string identifies a library, and the search is made over the libraries specified by these char-strings in left-to-right order. LI overrides any previous LI-command within the same module.

15.2 *LX (library external) command*

LX-command → "L" X-variable ("," char-string) + "."

The LX-command specifies the library or libraries from which the individual external reference corresponding to the X-variable is to be satisfied.

16. Noyau FUMOM

Les possibilités suivantes doivent être supportées par une réalisation conforme minimale.

Expressions:

Opérateurs: + , - , @NEG

(Implique une pile d'au moins deux valeurs)

Variables:

G,I,L,P,R,S,X (seuls W et N manquent)

Commandes:

MB

ME

SB

ST (sans attributs N ou P)

SA

NI,NX

LR,IR (sans options)

AS

17. Format binaire

Le format FUMOM est spécifié en vue de la portabilité sur divers supports. Un format binaire optionnel peut être réalisé pour un usage interne lorsque l'espace des fichiers ou les impératifs de temps pour les entrées/sorties sont critiques. Une réalisation du format binaire doit fournir une traduction bidirectionnelle entre celle-ci et sa forme caractère. Voir l'annexe B pour un exemple de codage binaire.

18. Bibliographie

[B1] FRASER, CHRISTOPHER W. et HANSON, DAVID R. A Machine-independent Linker. *Software - Practice and Experience*, vol. 12, 1982, pp 351-366.

[B2] MONTUELLE, Jean, CUFOM: The CERN Universal Format for Object Modules. *Rapport n° DD/78/21*. Genève, Suisse: CERN Data Handling Division, oct. 1978.

[B3] MONTUELLE, J. et WILLERS, I.L. Cross Software Using a Universal Object Module Format, CUFOM. *Proceedings of Euro IFIPB 79 - European Conference on Applied Information Technology*, SAMET, P.A. Ed. Amsterdam: North Holland Publishing Company, pp. 627-632.

[B4] PRESSER, Leon et WHITE, John R. Linkers and Loaders. *Computing Surveys*, vol. 4, n° 3, sept. 1972, pp. 149-157.

16. MUFOM kernel

The following features shall be supported by a minimally conforming implementation.

Expressions:

Operators: + , - , @NEG

(Implied stack at least two values deep)

Variables:

G,I,L,P,R,S,X (only W and N missing)

Commands:

MB

ME

SB

ST (without N or P attributes)

SA

NI,NX

LR,IR (without options)

AS

17. Binary format

The MUFOM format is specified for media portability. An optional binary format may be implemented for internal use where file space or I/O time is critical. An implementation of the binary format shall provide bidirectional translation between it and the character form. See Appendix B for an example of a binary encoding.

18. Bibliography

[B1] FRASER, CHRISTOPHER W. and HANSON, DAVID R. A Machine-independent Linker. *Software - Practice and Experience*, vol. 12, 1982, pp 351-366.

[B2] MONTUELLE, Jean, CUFOM: The CERN Universal Format for Object Modules. *Report No. DD/78/21*. Geneva, Switzerland: CERN Data Handling Division, Oct. 1978.

[B3] MONTUELLE, J. and WILLERS, I.L. Cross Software Using a Universal Object Module Format, CUFOM. *Proceedings of Euro IFIPB 79 - European Conference on Applied Information Technology*, SAMET, P.A. Ed. Amsterdam: North Holland Publishing Company, pp. 627-632.

[B4] PRESSER, Leon and WHITE, John R. Linkers and Loaders. *Computing Surveys*, vol. 4, No. 3, Sept. 1972, pp. 149-157.

ANNEXE A (informative)

SYNTAXE

variable → lettre (lettre | digit) *

digit → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

lettre-hexadécimale → "A" | "B" | "C" | "D" | "E" | "F"

digit-hexadécimal → digit | lettre-hexadécimale

nombre-hexadécimal → digit-hexadécimal +

lettre-non-hexadécimale → "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

lettre → lettre-hexadécimale | lettre-non hexadécimale

alphanumérique → lettre | digit

identificateur → lettre alphanumérique *

caractère → tout caractère ASCII

longueur-chaîne-caractère → digit-hexadécimal digit-hexadécimal

chaîne-caractère → longueur-chaîne-caractère caractère *

variable-FUMOM → lettre-non-hexadécimale nombre-hexadécimal?

expression → nombre-hexadécimal | variable-FUMOM | expression-polonaise | expression-conditionnelle

expression-polonaise → (opérande ",") * fonction

opérande → expression

fonction → "@" identificateur | opérateur

opérateur → "+" | "-" | "/" | "*" | "<" | ">" | "=" | "#"

expression-polonaise → "@F"

expression-polonaise → "@T"

expression-polonaise → opérande "," f-monad

f-monad → "@ABS" | "@NEG" | "@NOT" | "@ISDEF"

expression-polonaise → opérande1 "," opérande2 "," f-dyad

APPENDIX A (informative)

COLLECTED SYNTAX

variable → letter (letter | digit) *

digit → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

hexletter → "A" | "B" | "C" | "D" | "E" | "F"

hexdigit → digit | hexletter

hexnumber → hexdigit +

nonhexletter → "G" | "H" | "I" | "J" | "K" | "L" | "M" |
"N" | "O" | "P" | "Q" | "R" | "S" | "T" |
"U" | "V" | "W" | "X" | "Y" | "Z"

letter → hexletter | nonhexletter

alphanum → letter | digit

identifier → letter alphanum *

character → any ASCII graphic character

char-string-length → hexdigit hexdigit

char-string → char-string-length character *

MUFOM-variable → nonhexletter hexnumber?

expression → hexnumber | MUFOM-variable |
polish-expr | conditional-expr

polish-expr → (operand ",") * function

operand → expression

function → "@" identifier | operator

operator → "+" | "-" | "/" | "*" | "<" | ">" | "=" | "#"

polish-expr → "@F"

polish-expr → "@T"

polish-expr → operand "," monad-f

monad-f → "@ABS" | "@NEG" | "@NOT" | "@ISDEF"

polish-expr → operand1 "," operand2 "," dyad-f

f-dyad → "+" | "-" | "/" | "*" | "@MAX" | "@MIN" | "@MOD" |
"<" | ">" | "=" | "#" | "@AND" | "@OR" | "@XOR"

expression-polonaise → opérande1 "," opérande2 ","
opérande3 "," "@EXT"

expression-polonaise → opérande1 "," opérande2 ","
opérande3 "," opérande4 "," "@INS"

expression-polonaise → valeur "," condition "," numerreur "," "@ERR"

valeur → expression

numerreur → expression

expression-conditionnelle → condition "," "@IF" "," expression1 ","
"@ELSE" "," expression2 "," "@END"

condition → expression

variable-FUMOM → lettre-non-hexadécimale nombre-hexadécimal?

variable-G → "G"

variable-I → "I" nombre-hexadécimal

variable-L → "L" numéro-section?

variable-N → "N" nombre-hexadécimal

variable-P → "P" numéro-section?

variable-R → "R" numéro-section?

variable-S → "S" numéro-section?

numéro-section → nombre-hexadécimal

variable-W → "W" nombre-hexadécimal

variable-X → "X" nombre-hexadécimal

commande-MB → "MB" configuration-machine-cible ("," nom-module)? "."

configuration-machine-cible → identificateur

nom-module → chaîne-caractère

commande-ME → "ME."

commande-DT → "DT" digit * "."

commande-AD → "AD" bits-par-UMA ("," UMA-par-adresse
("," ordre)?)? "."

```
dyad-f →    "+" | "-" | "/" | "*" | "@MAX" | "@MIN" | "@MOD" |
            "<" | ">" | "=" | "#" | "@AND" | "@OR" | "@XOR"

polish-expr →       operand1 "," operand2 ","
                    operand3 "," "@EXT"

polish-expr →       operand1 "," operand2 ","
                    operand3 "," operand4 "," "@INS"

polish-expr → value "," condition "," errornum "," "@ERR"

value → expression

errornum → expression

conditional-expr →  condition "," "@IF" "," expression1 "," "@ELSE" "," expression2 "," "@END"

condition → expression

MUFOM-variable → nonhexletter  hexnumber?

G-variable → "G"

I-variable → "I" hexnumber

L-variable → "L" section-number?

N-variable → "N" hexnumber

P-variable → "P" section-number?

R-variable → "R" section-number?

S-variable → "S" section-number?

section-number → hexnumber

W-variable → "W" hexnumber

X-variable → "X" hexnumber

MB-command → "MB" target-machine-configuration ("," module-name)? "."

target-machine configuration → identifier

module-name → char-string

ME-command → "ME."

DT-command → "DT" digit * "."

AD-command →  "AD" bits-per-MAU ("," MAUs-per-address
              ("," order)? )? "."
```

bits-par-UMA → nombre-hexadécimal

UMA-par-adresse → nombre-hexadécimal

ordre → "L" | "M"

commande-CO → "CO" niveau-commentaire? "," texte-commentaire "."

niveau-commentaire → nombre-hexadécimal

texte-commentaire → chaîne-caractère

commande-CS → "CS" somme-de-contrôle? "."

somme-de-contrôle → digit-hexadécimal digit-hexadécimal

commande-SB → "SB" numéro-section "."

commande-ST → "ST" numéro-section ("," type-section) *
("," nom-section)? "."

type-section → lettre

nom-section → chaîne-caractère

commande-SA → "SA" numéro-section "," frontière-UMA?
("," taille-page)? "."

frontière-UMA → expression

taille-page → expression

commande-NI → "N" variable-I "," nom-def-ext "."

nom-def-ext → chaîne-caractère

commande-NX → "N" variable-X "," nom-ref-ext "."

nom-ref-ext → chaîne-caractère

commande-NN → "N" variable-N "," nom "."

nom → chaîne-caractère

commande-AT → "AT" variable "," entrée-table-type ("," niveau-lex
("," nombre-hexadécimal) *)? "."

variable → variable-I | variable-N | variable-X

entrée-table-type → nombre-hexadécimal

niveau-lex → nombre-hexadécimal

commande-TY → "TY" entrée-table-type ("," paramètre) + "."