



ASSEMBLE

Brainstorm © 1993

Mode d'emploi

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part que les "copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective", et, d'autre part, que les analyses et courtes citations dans un but d'exemple et d'illustration, "toute représentation ou reproduction intégrale ou partielle, faite sans le consentement des auteurs ou de leurs ayants droit ou ayants cause, est illicite" (alinéa 1 de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

Les marques citées sont des marques déposées par leurs propriétaires respectifs.

Brainstorm n'offre aucune garantie, expresse ou tacite, concernant ce logiciel et ne pourra être tenu responsable des préjudices éventuels dus à son utilisation.

Table des matières

Table des matières	2
Section I • Introduction	7
Copie de sauvegarde	7
Installation	7
Conventions	7
Appeler l'assembleur	8
Ligne de commande	8
Types de fichiers	9
Section II • Syntaxe de l'assembleur	10
Généralités	10
Ligne de commentaire	10
Ligne de source	10
Label	10
Labels locaux	11
Opération	11
Opérande	12
Commentaire	12
Variables, constantes et expressions	13
Variables	13
Constantes	14
Expressions	15
Expressions absolues	15
Expressions relatives (relogables)	15
Opérateurs reconnus par l'évaluateur	15
Opérateurs arithmétiques	15
Opérateurs de décalage	15
Opérateurs logiques	16
Opérateurs flottants	16
Opérandes de FPUROM	17
Priorités des opérateurs	18
Combinaisons d'opérateurs	18
Modes d'adressages	19
Sections	22
Section III • Directives	23
Introduction	23
Alignement modulo 16 bits	23
Contrôle d'assemblage	23
OPT	23
@<#>	23
B+/-	23
C+/-	23
D+/-	23
E+/-	24

I<chemin>	24
L0 L1 L2	24
M+/-	24
O[<#>]+/-	24
OW[<#>]+/-	24
O=<nom>	24
P<nom>	25
P=<processeur>	25
S+/-	25
V+/-	25
W+/-	25
X+/-	25
Y+/-	25
FOPT	26
INCLUDE	26
INCDIR	27
INCBIN	27
OUTPUT	27
EVEN	28
CNOP	28
ORG	29
SECTION	29
TEXT	29
DATA	29
BSS	29
OFFSET	29
END	30
FAIL	30
ERROR	30
WARNING	30
Déclaration de variables	31
EQU	31
SET	31
EQUR	32
REG	32
Déclaration de variables pour virgule flottante	33
FEQU	33
FSET	33
FREG	34
FEQUR	34
Déclaration de données initialisées	35
DC	35
DCB	35
ASCII	36
ASCIIZ	36
ASCIIL	36
Déclaration de données non initialisées	37

DS	37
Section RS	37
Introduction	37
RS	37
RSSET	38
RSRESET	38
RSEVEN	38
RSSSTRUCT	38
RSUNION	39
RSEND	39
Déclaration de paramètres	40
CARGS	40
PARGS	40
Contrôle du listing	41
LIST	41
NOLIST	41
SPC	41
TTL	41
SUBTTL	41
PAGE	42
NOPAGE	42
LLEN	42
PLEN	42
LISTCHAR	42
TABS	42
Directives externes	43
XREF	43
XDEF	43
Directives diverses	44
SUPER	44
USER	44
MC68000	44
MC68010	44
MC68020	44
MC68030	45
MC68EC030	45
MC68040	45
MC68EC040	45
MCCPU32	46
MC68881	46
MC68851	46
Macros	47
Introduction	47
Vue générale	47
Définir une macro	47
En-tête	47
MACRO	47

Corps	47
SHIFTM	48
Fin de déclaration.....	48
ENDM	48
Fin d'expansion	48
EXITM	48
Paramètres	49
Appel.....	49
Expansion du corps.....	50
Assemblage conditionnel	53
Introduction	53
En-tête	53
IF	54
IFEQ	54
IFGT.....	54
IFGE	54
IFLT	54
IFLE	54
IFC	55
IFNC.....	55
IFD.....	55
IFND	55
Alternative.....	55
ELSE	55
Fin	56
ENDIF	56
Assemblage répété.....	57
REPT.....	57
ENDR	57
Appendices.....	58
Appendice A • Variables réservées	58
-_ASM	58
-_DBG	58
-_FPU.....	58
-_LK.....	58
-_MMU	58
-_MPU	59
-_RS.....	59
NARG	59
Appendice B • Optimisations	60
Appendice C • Directives	61
Appendice D • Instructions.....	62
Introduction	62
Instructions MPU	62
Instructions MMU	66
Instructions FPU	67
Appendice E • Messages d'erreur	69

Appendice F • Messages d'avertissement	76
Optimisations	76
Divers.....	78
Appendice G • Spécificités Atari.....	79
Formats de sortie	79
Gestion des sections	79
Compatibilité avec d'autres assembleurs	80
Devpac (Hisoft) :.....	80
Metacomco Assembler (Metacomco)	80
Pure Assembler (Pure Software)	80
MadMAC (Atari Corporation)	81
Gfa Assembleur (Micro-Application)	81
Profimat ST (Micro-Application).....	82
Turbo-Assembler (Sigma Software)	82
GNU Assembler (GNU Software)	82
Appendice H • Bibliographie	83
Française	83
Anglaise.....	83
Appendice I • Table ASCII	84
Appendice J • Index.....	85

Section I • Introduction

Ce manuel a pour objet d'apprendre à se servir d'Assemble 68xxx, un assembleur pour la famille 68xxx. Celui-ci permet l'assemblage rapide de sources en assembleur et inclut toutes les options souhaitables d'optimisation, d'assemblage conditionnel et répété, ainsi que de débogage.

Copie de sauvegarde

La disquette jointe à ce manuel est votre original d'ASM. Avant toute manipulation, faites-en une copie, ou installez-le directement sur votre disque dur. Gardez l'original dans un endroit sûr où vous pourrez le retrouver si besoin est.

Avec ce manuel, vous trouverez aussi une carte d'enregistrement de client. Remplissez-la et renvoyez-la-nous le plus vite possible. Vous pourrez dès lors bénéficier des mises à jour et être avertis des nouveaux produits de Brainstorm!

Installation

Pour installer **ASM** sur disque dur, il suffit de créer un répertoire (nommé par exemple **ASM**) et de copier le contenu de la disquette à l'intérieur. Pour les utilisateurs de shells comme **Craft** par exemple, il est conseillé de mettre **asm.ttp** dans le **path**, soit en ajoutant explicitement à la variable **PATH** son chemin d'accès, soit en le copiant dans **BIN**.

Conventions

Le symbole désigne un exemple.

Le symbole désigne une remarque simple.

Le symbole désigne une remarque importante.

<> entoure un symbole obligatoire.

[] entoure un symbole optionnel.

| indique un choix parmi plusieurs symboles.

Espace blanc : indique toute combinaison des caractères espace et tabulation.

Label : se traduit en français par **étiquette**. Puisque le "langage informatique" utilise le mot label plutôt qu'étiquette, nous en ferons autant dans ce manuel.

Appeler l'assembleur

Ligne de commande	
Syntaxe : <code>asm [-<opt>, ...] <nom du source></code>	
où les options sont (voir la directive OPT pour plus de détails)	
Option	Explication
<code>@<#></code>	Nombre de chiffres pour \@
<code>B+/-</code>	Ne pas générer de fichier binaire
<code>C+/-</code>	Distinguer les labels en majuscule/ minuscule
<code>D+/-</code>	Générer des symboles de 8 caractères (DRI)
<code>D=<exp></code>	Définir un EQU
<code>E+/-</code>	Générer les EQU dans les symboles
<code>I<nom></code>	Répertoire d'include (voir INCDIR)
<code>LO/1/2</code>	Choix du type de fichier binaire de sortie
<code>M+/-</code>	Inclure l'expansion de MACRO dans le listing
<code>N+/-</code>	Economiser de la mémoire
<code>O<#>+/-</code>	Mode d'optimisation (1<#<15)
<code>OW+/-</code>	Avertir en cas d'optimisation
<code>O=<nom></code>	Nom du fichier à générer (voir OUTPUT)
<code>P<nom></code>	Nom du listing à générer (voir LIST)
<code>P=<mcu></code>	Selectionner le processeur (voir MCXXX)
<code>R+/-</code>	Allouer la mémoire par bloc
<code>S+/-</code>	Lister la table des symboles
<code>U+/-</code>	Interdire les instructions privilégiées
<code>V+/-</code>	Afficher les statistiques en fin d'assemblage
<code>W+/-</code>	Afficher les avertissements
<code>X+/-</code>	Générer des symboles de 22 caractères
<code>Y+/-</code>	Générer le débogage source (compatible Pure C)

Si + et - sont omis après une option, + est pris par défaut.

Les options grisées doivent être définies avant le début de l'assemblage et ne peuvent plus l'être par la suite.

Lors de l'assemblage, ASM affiche ses messages (erreurs et avertissements) à l'écran. Le défilement peut en être arrêté avec **Ctrl-S**, et repris avec **Ctrl-Q**. L'assemblage peut être définitivement abandonné avec **Ctrl-C**.

Le format des messages est le suivant :

`<?><n°> File <noms de fichier> Line <n° de ligne> (Message)
[Ligne du source en cause]`

`<?>` peut être **E** s'il s'agit d'une erreur ou **W** s'il s'agit d'un avertissement.

Types de fichiers

ASM peut générer 4 types de fichiers binaires :

- **programme**, pour l'ATARI (mode par défaut).
- **objet Pure C**, pour être linké.
- **objet DRI**, pour être linké.
- **absolu**, avec ORG, généralement destiné à être mis en **EPROM**.

Le bureau de l'ATARI connaît 5 extensions pour les programmes :

- **PRG**, qui indique un programme utilisant le GEM.
- **APP**, comme ci-dessus.
- **GTP**, comme ci-dessus mais avec passage de paramètres.
- **TOS**, qui indique un programme n'utilisant pas le GEM.
- **TTP**, comme ci-dessus mais avec passage de paramètres.

Un autre suffixe, **ACC**, est utilisé pour les programmes devant être chargés au lancement du bureau, et gardés en mémoire. Ils apparaissent généralement dans la barre de menu de gauche.

Le suffixe par défaut en mode programme ATARI est **PRG**. En mode Pure C et DRI, celui-ci est **O**.

Le nom du fichier sans suffixe est celui du source principal (le premier à être assemblé).

Si la directive OUTPUT est utilisée, plusieurs choix sont possibles :

- le nom complet (avec extension) est indiqué. Il sera pris tel quel comme nom du fichier généré.
- le nom sans extension est indiqué. Dans ce cas, l'extension par défaut lui sera rajoutée.
- l'extension seule est indiquée. Le nom du fichier source (sans extension) lui sera alors ajouté pour obtenir le nom du fichier généré.

Le choix du type de fichier de sortie se fait par la ligne de commande ou par la directive OPT (voir celle-ci). Ce choix doit néanmoins se faire avant tout assemblage de code, car il entraîne un comportement différent d'ASM.

Section II • Syntaxe de l'assembleur

Généralités

Un source est constitué d'un ensemble de lignes, chacune pouvant être :

- une ligne vide
- un commentaire
- une instruction
- une directive
- un appel de macro

ASM interdit les lignes supérieures à 256 caractères.

Ligne de commentaire

Une ligne de commentaire contient comme premier caractère différent d'un espace blanc le caractère '*' ou ';'.

Ligne de source

Elle est constituée des champs suivants :

- label (optionnel)
- mnémonique (instruction ou directive) (obligatoire)
- un ou plusieurs opérandes (optionnels suivant l'opération)
- commentaire (optionnel)

[label]	mnémonique	opérandes	[commentaire]
foo:	move.l	d0,d1	;commentaire

Label

Il doit être le premier champ de la ligne. Un label commençant sur la première colonne de texte peut être terminé par un espace blanc, un retour chariot ou un ':'. Un label ne commençant pas sur la première colonne **doit** être terminé par un ':'. Dans tous les cas, le caractère terminal ne fait pas partie du nom du label.



```

foo:           nop
foo:nop       nop
foo            nop
              foo: nop
              foo:nop

```

Si ce label ne précède pas une des directives suivantes : EQU, SET, REG, EQUR, FEQU, FSET, FREG, FEQUR, MACRO, sa valeur est égale à celle du compteur courant d'assemblage au **début** de la ligne, avec **toutes** les autres directives ou instructions.

☞ Nous conseillons de mettre un ':' après les variables ne précédant pas les directives susnommées, et de ne pas en mettre après les autres. Cela facilite la lecture du source. Nous déconseillons aussi de mettre des points dans un label, car cela est en principe réservé aux noms d'éléments de structure.

Labels locaux

Les labels locaux se définissent comme les autres labels, à part le fait qu'ils sont précédés par le caractère point ('.'). Ils sont accessibles jusqu'au label global suivant. Une suite de chiffres terminée par un caractère dollar ('\$') est aussi un label local. Nous déconseillons toutefois l'utilisation de cette syntaxe, qui n'est comprise que par compatibilité avec d'autres assembleurs.

Opération

Ce champ suit le champ label et en est séparé par un espace blanc. Il ne peut commencer sur la première colonne. Il contient l'une des entités suivantes :

- un **mnémonique** d'instruction
- une **directive** d'assemblage
- un nom de **macro**, symbolisant l'appel à cette macro.

Les mnémoniques et directives sont reconnus en majuscule ou en minuscule (pas les noms de macro, bien sûr). Nous conseillons néanmoins d'écrire les directives en majuscule et les mnémoniques en minuscule afin d'améliorer la lisibilité du source. Dans beaucoup de cas, il est possible de spécifier une taille en plus de l'opération, séparée de cette dernière par un point (.).

Les tailles acceptées sont les suivantes :

Lettre	Nom	Taille	MCU
B	Byte (ou S pour les Bcc)	8 bits	MPU
W	Word (taille par défaut)	16 bits	MPU
L	Long word	32 bits	MPU
S	Single precision	32 bits	FPU
D	Double precision	64 bits	FPU
X	Extended precision	80 bits	FPU
P	Packed decimal	80 bits	FPU
Q	Quad word (double long)	64 bits	MMU

Opérande

S'il est présent, ce champ suit le champ d'opération et en est séparé par un espace blanc. Si plusieurs **opérandes** sont nécessaires, elles sont séparées entre elles par des virgules ','. Ces dernières peuvent être précédées ou suivies par un espace blanc. Il est en revanche impossible de mettre un espace blanc au milieu d'une opérande, car la fin de la ligne suivant cet espace blanc serait prise comme un commentaire.

Voir la section MODES D'ADRESSAGE pour connaître l'ensemble des modes d'adressage.

Commentaire

Ceci est le dernier champ (optionnel) de la ligne et suit le dernier champ opérande. Il en est séparé par un symbole de **commentaire**, ';' ou '**', ou par un espace blanc. Nous recommandons toutefois l'utilisation du ';' afin d'améliorer la clarté du source et d'augmenter la vitesse d'assemblage en levant les ambiguïtés grammaticales.

Si un symbole de commentaire est dédoublé (;; et **) dans le corps d'une macro, il n'est pas mémorisé. Cela permet d'économiser de la mémoire et d'augmenter la vitesse d'assemblage.

☞ Si l'étoile ('*') est utilisée comme symbole de commentaire, elle doit être **obligatoirement** précédée d'un espace blanc (sauf en début de ligne).

Variables, constantes et expressions

Variables

Les noms de variables peuvent atteindre la taille maximum d'une ligne (256 caractères). Ils peuvent comprendre des caractères alphabétiques (de a à z et de A à Z), des chiffres (de 0 à 9), ainsi que les caractères souligné (_), point d'interrogation (?) et point (.). Le premier caractère ne peut être un chiffre.

☞ Nous déconseillons toutefois l'utilisation du caractère '.', celui-ci entraînant des ambiguïtés syntaxiques comme :

tst.w

sera bien reconnu comme un label, mais

tst.w

sera évidemment pris comme une instruction!

De plus, les éléments de structures et d'unions sont systématiquement référencés par <nom de la structure / union>.<nom de l'élément>. On évite bien des confusions en restreignant l'usage du point à cette seule syntaxe.

Les noms de variables ne peuvent être identiques aux symboles réservés reconnus par ASM (majuscules ou minuscules indifférent) :

D0 à D7, A0 à A7, SP, PC, USP, SR, CCR, ZA0 à ZA7, ZPC

FPO à FP7, FPCR, FPSR, FPIAR

AC0, AC1, ACUSR, CAAR, CACR, DACR0, DACR1, DFC, DTT0, DTT1, IACR0, IACR1, ISP, ITT0, ITT1, MSP, SFC, TTO, TT1, VBR

AC, BAD, BAC, CAL, CRP, DRP, PCSR, MMUSR, PMMUSR, PSR, SCC, SRP, TC, URP, VAL

Les noms de macro sont légèrement différents, puisqu'ils ne sont pas évalués. Ils peuvent donc commencer par un chiffre, et ne peuvent être identiques aux mnémoniques et directives reconnus par ASM (vous en serez immédiatement avertis, de toute façon). Par ailleurs, ils ne peuvent contenir le caractère point (.).

Constantes

Les constantes peuvent être exprimées dans quatre bases :

Nom	Préfixe	Exemple
décimal	sans	256
hexadécimal	\$	\$100 (décimal 256)
binaire	%	%10000000 (décimal 256)
octal	@	@400 (décimal 256)

et sous forme de chaîne alphanumérique encadrée d'**apostrophes** ou de **guillemets**.

- ☞ 'abcd', "aaaa", "zzzz"
- 'voilà un guillemet: "'
- "voilà une apostrophe: ""

☞ Il faut encadrer du texte contenant des guillemets par des apostrophes, et réciproquement. On peut aussi échapper un guillemet ou une apostrophe en le doublant.

- ☞ """ab"""; donne "ab"
- ""cd""; donne 'cd'

Les constantes chaînes ne peuvent dépasser quatre caractères, sauf dans le cas des directives DC.B, ASCII, ASCIIZ, ASCIIL. Si la chaîne est inférieure à quatre caractères, elle est calée à droite et complétée à gauche par des zéros.

Les constantes à virgule flottante peuvent être exprimées de deux manières :

- en hexadécimal, avec comme préfixe '\$' ou ':' (notation Motorola)
- en notation flottante classique, c'd un nombre en décimal constituant la partie entière, pouvant être suivie par un point (point décimal) et un nombre en décimal représentant la partie fractionnelle, éventuellement suivie elle-même par un nombre en décimal constituant l'exposant, positif ou négatif, précédé du caractère e ou E. Ce nombre flottant peut éventuellement être précédé d'un moins unaire ('-').

☞ Les constantes en notation flottante ainsi que tout calcul flottant demandent un coprocesseur arithmétique.

Expressions

Les expressions arithmétiques peuvent contenir un ou plusieurs symboles et des opérateurs unaires ou binaires. Les symboles contenus dans les expressions peuvent être :

- des variables, possédant une valeur absolue ou relative
- des nombres, possédant une valeur absolue
- un symbole spécial, l'étoile (*), possédant comme valeur relative celle du compteur d'assemblage courant au début de la ligne en cours d'assemblage.

A la fin de l'évaluation de l'expression, le résultat peut être rangé en deux catégories :

- absolu, donc indépendant de l'adresse de début du programme
- relogable, c'est à dire un offset par rapport au début du programme

Expressions absolues

Une expression peut être absolue parce qu'elle n'est composée que de nombres, de symboles absous ou de différences de symboles relatifs.

Expressions relatives (relogables)

Elles sont de la forme :

- relogable + absolu
- relogable - absolu
- absolu + relogable

Opérateurs reconnus par l'évaluateur**Opérateurs arithmétiques**

Symbol	Explication
+	addition
-	soustraction
*	multiplication
/	division
+	plus unaire
-	moins unaire = NEG (complément à 2)

Opérateurs de décalage

Symbol	Explication
>>	décalage vers la droite
<<	décalage vers la gauche

Opérateurs logiques

Symbole	Explication
&	et
!,	ou
^	ou exclusif
~	non unaire = NOT (complément à 1)
==, =	égal
!=, <>	non égal
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal

Opérateurs flottants

Opérande	Opcode	Explication
ABS	FABS	Valeur absolue(x)
ACOS	FACOS	Arc cosinus(x)
ASIN	FASIN	Arc sinus(x)
ATAN	FATAN	Arc tangente(x)
ATANH	FATANH	Arc tangente hyperbolique(x)
COS	FCOS	Cosinus(x)
COSH	FCOSH	Cosinus hyperbolique(x)
ETOX	FETOX	Exponentielle(x)
ETOXM1	FETOXM1	Exponentielle(x)-1
EXP	FGETEXP	Exposant(x)
FPUROM	FMOVECR	Constante flottante ROM(x)
INT	FINT	Partie entière(x)
INTRZ	FINTRZ	Partie entière(x) vers 0
LOG10	FLOG10	Log en base 10(x)
LOG2	FLOG2	Log en base 2(x)
LOGN	FLOGN	Log népérien(x)
LOGNP1	FLOGNP1	Log népérien(x+1)
MAN	FGETMAN	Mantisse(x)
MOD	FMOD	Modulo(x,y)
REM	FREM	Partie restante(x,y)
SCALE	FSCALE	Additionne exposant(x,y)
SIN	FSIN	Sinus(x)
SINH	FSINH	Sinus hyperbolique(x)
SQRT	FSQRT	Racine carrée(x)
TAN	FTAN	Tangente(x)
TANH	FTANH	Tangente hyperbolique(x)
TENTOX	FTENTOX	10^x
TWOTOX	FTWOTOX	2^x

FPUROM permet d'accéder aux constantes flottantes présentes dans la ROM du 68881/2.

Opérandes de FPUROM

Opérande	Résultat
0	π
11	$\text{Log10}(2)$
12	e
13	$\text{Log2}(e)$
14	$\text{Log10}(e)$
15	0.0
48	$\ln(2)$
49	$\ln(10)$
50	10^0
51	10^1
52	10^2
53	10^4
54	10^8
55	10^{16}
56	10^{32}
57	10^{64}
58	10^{128}
59	10^{256}
60	10^{512}
61	10^{1024}
62	10^{2048}
63	10^{4096}

Priorités des opérateurs

Plus prioritaire	0
	Plus, moins, non unaire
	Opérateurs de décalage
	Et, ou, ou exclusif
	Multiplication, division
	Addition, soustraction
Moins prioritaire	Egalité, inégalité, inférieur, supérieur, inférieur ou égal, supérieur ou égal

Combinaisons d'opérateurs

	A-A	A-R	R-A	R-R
Décalages	A	•	•	•
Logiques	A	•	•	•
Multiplication	A	•	•	•
Division	A	•	•	•
Soustraction	A	•	R	A
Addition	A	R	R	•
Comparaisons	A	A	A	A

- ☞ Toutes les opérations sont réalisées sur un long de 32 bits signé (en Extended pour les calculs en virgule flottante). Tout dépassement entraîne une erreur d'évaluation.
- ☞ Les calculs flottants nécessitent un coprocesseur arithmétique.

Modes d'adressages

Les noms des registres sont compris indifféremment en majuscule ou minuscule. SP peut être utilisé à la place de A7 (pour une meilleure lisibilité).

Le 68000-68010 permet 7 modes d'adressage différents : (Sz signifie Size, càd la taille du registre, Sc signifie Scale, càd l'échelle du registre)

- **Dn** Data register

Registres de donnée D0 à D7.

move d0,d1

- **An** Address register

Registres d'adresse A0 à A7.

movea d0,a0

- **(An)** Indirect

Contenu de registre d'adresse

move (a0),d0

- **(An)+** Indirect postincremented

Contenu de registre d'adresse avec post-incrémantion

move (a0)+,d0

- **-(An)** Indirect preincremented

Contenu de registre d'adresse avec pré-décrémentation

move -(a0),d0

- **d16(An)** Indirect with 16-bit displacement

Déplacement sur 16 bits additionné au registre d'adresse An.

move 20(a0),d0

- **d16(PC)** Indirect with 16-bit displacement

Déplacement sur 16 bits additionné à la valeur du PC au moment de l'exécution. Le déplacement doit être un label (ne peut donc être absolu).

move label(pc),d0

- **d8(An,XnSz)** Indirect with 8-bit displacement and Index register

Déplacement sur 8 bit additionné au registre d'adresse An et au registre de donnée ou d'adresse Xn. Sz est la taille de Xn, Word ou Long.



move 4(a0,d0),d0

- **d8(PC,XnSz)** Indirect with 8-bit displacement and Index register

Déplacement sur 8 bit additionné à la valeur du PC au moment de l'exécution et au registre de donnée ou d'adresse Xn. Sz est la taille de Xn, Word ou Long. Le déplacement doit être un label.



move label(pc,d0),d0

- **#i** Immediate

Valeur immédiate, dont la taille est déterminée par celle de l'instruction.



move #4,d0

- **abs.w** Absolute short

Adresse absolue sur 16 bits.



move \$100.w,d0

- **abs.l** Absolute long

Adresse absolue sur 32 bits.



move label,d0

A partir du 68020, le mode d8(An,XnSz) s'est vu ajouté un coefficient pour le registre Xn, appelé Scale :

- **d8(An,XnSz*Sc)** et **d8(PC,XnSz*Sc)**, où Sc (Scale) peut prendre les valeurs 1 (identique au 68000), 2, 4 et 8.



move 4(a0,d0*4),d0

move label(pc,d0.w*2),d0

Ont été aussi ajoutés trois modes :

- **(BdSz,An,XnSz*Sc)** et **(BdSz,PC,XnSz*Sc)**

Base displacement.

Bd est le **base displacement**, offset ajouté au registre An (ou au PC courant) et au registre Xn. Le Bd peut prendre Word ou Long comme taille (par défaut Long). Si la taille spécifiée est le Byte, alors cela revient à écrire d8(AnXn), ce qui est un mode 68000. Si l'on veut supprimer le registre d'adresse, on peut soit l'omettre simplement, soit mettre ZAn (n compris entre 0 et 7).

Pour supprimer le PC, il faut indiquer ZPC.

On peut supprimer l'un quelconque des composants du mode d'adressage, ce qui donne les combinaisons suivantes :

(BdSz,XnSzSc) équivaut à (BdSz,ZA0,XnSzSc)

(BdSz) équivaut à (BdSz,ZA0)

(XnSzSc) équivaut à (ZA0,XnSzSc)



```
move    (4,a0,d0*4),d0
move    (label,pc,d0.w*2),d0
```

☞ Attention toutefois au pièges suivants :

- (Bd) sans taille précisée donnera un mode abs. Il faut donc soit indiquer la taille du Bd, soit écrire (Bd,ZA0).
- (An) en tant que (Xn) donnera un mode (An) (contenu de registre d'adresse). Il faut donc indiquer la taille et/ou le scale, ou bien écrire (ZA0,An).
- il est possible d'accéder au contenu d'un registre de donnée comme dans le mode (An), en écrivant (Dn). Mais comme la taille par défaut du registre Xn est le Word, il faut préciser la taille Long et écrire (Dn.l) pour arriver au même résultat.

- ([Bd,An,XnSzSc],od) et ([Bd,PC,XnSzSC],od)
ou ([Bd,An,XnSzSc]) et ([Bd,PC,XnSzSC])

Outer displacement preindexed.

Od est l'**outer displacement**, ou déplacement extérieur. L'expression entre crochets est identique au paragraphe précédent. Une fois l'adresse calculée, son contenu (en Long) sera pris comme adresse de base. A cette dernière est ajoutée l'Od, si présente.



```
move    ([4,a0,d0*4],label),d0
move    ([pc,d0.w*2]),d0
```

- ([Bd,An],XnSzSc,od) et ([Bd,PC],XnSzSc,od)
ou ([Bd,An],XnSzSc) et ([Bd,PC],XnSzSc)

Outer displacement postindexed

De même qu'au paragraphe précédent, l'expression entre crochets est calculée et son contenu (en long) pris comme adresse de base. Lui est alors ajouté le registre d'index selon les règles précédemment évoquées, puis l'Od si présente.



```
move    ([4,a0],d0*4),d0
move    ([pc],d0.w*2,$12),d0
```

☞ Si la taille du registre d'index est omise, la taille par défaut est le Word.

Les modes d8(An,Xn) et d8(PC,Xn) peuvent s'écrire sans le d8. Il est alors automatiquement considéré à 0 :
(A0,D7,W*4) est équivalent à 0(A0,D7,W*4)

Sections

ASM reconnaît trois types de sections logiques :

La section **TEXT** (texte) est censée ne contenir que du code, càd des instructions. Elle peut évidemment contenir aussi des données.

La section **DATA** (donnée) est censée ne contenir que des données. Elle peut aussi contenir du code.

La section **BSS** (Block Storage Segment) ne peut contenir que des données non initialisées, càd initialisées au lancement du programme à 0.

Ces trois sections peuvent être déclarées par la directive SECTION suivie de TEXT, DATA ou BSS ou par les directives TEXT, DATA ou BSS.

ASM reconnaît aussi une autre section appelée **OFFSET**. Elle est destinée à définir des valeurs d'offset servant généralement en index de registre d'adresse.

Section III • Directives

Introduction

Les directives servent à demander à ASM d'effectuer certaines actions au cours de l'assemblage. Elles ne sont pas effectivement assemblées, comme le sont les mnémoniques. Certaines d'entre elles demandent une déclaration de label préalable pour fonctionner. C'est le cas de : EQU, SET, REG, EQUR, FEQU, FSET, FREG, FEQUR, MACRO. Toutes les autres directives peuvent ou non être précédées d'un label, ce dernier étant finalement défini comme suivant le dernier mnémonique assemblé.

Alignement modulo 16 bits

Les instructions 68xxx doivent être alignées au word (16 bits) pour pouvoir être exécutées. De ce fait, ASM force cet alignement avant tout assemblage (sauf pour DC.B et DS.B, devant lesquels il faut explicitement utiliser EVEN ou CNOP pour obtenir l'alignement voulu).

☞ Les variables éventuellement définies devant l'instruction ne seront pas alignées.

Contrôle d'assemblage

OPT

Syntaxe : OPT <opt>[. <opt>...]
 <opt> peut être :

@<#>

Sert à régler le nombre de chiffres utilisés pour générer les labels uniques dans les macros avec \@.

Défaut : @3

B+/-

Demande à ASM de ne pas générer de fichier binaire.

Défaut : B-

C+/-

Autorise ou interdit la différenciation majuscule / minuscule des labels.

Défaut : C+

D+/-

Demande la génération des symboles de débogage **DRI** (8 caractères) dans l'exécutable.

Défaut : D-

E+/-

Demande la génération des EQUs dans les symboles de débogage.

Défaut : E+

I<chemin>

Fixe le répertoire d'**include**.

Voir aussi : INCDIR

Défaut : chemin courant

L0 L1 L2

Sélectionne le type de fichier exécutable :

L0 signifie exécutable.

L1 signifie format objet **Pure C**.

L2 signifie format objet **DRI**.

Cette option doit être sélectionnée avant toute génération de binaire et avant toute déclaration de XDEF et XREF.

Défaut : L0

M+/-

Autorise ou interdit la présence des appels de macro dans le listing.

Défaut : M-

O[<#>]+/-

Autorise ou interdit une **optimisation**. <#> est le numéro d'optimisation. S'il est omis, toutes les optimisations sont concernées. (Voir Appendice B, tableau des optimisations).

Défaut : O-

OW[<#>]+/-

Autorise ou interdit l'avertissement en cas d'optimisation. <#> est le numéro d'optimisation. S'il est omis, toutes les optimisations sont concernées. (Voir Appendice B, tableau des optimisations).

Défaut : OW-

O=<nom>

Fixe le nom du fichier à générer.

Voir aussi : OUTPUT

Défaut : nom du source principal avec l'extension **O** en cas d'objet ou **PRG** en cas d'exécutable.

P<nom>

Permet de spécifier un nom de fichier pour le **listing**. Par défaut, le listing est généré à l'écran (ce qui permet les indirections). Pour envoyer le listing sur le port parallèle, il suffit de mettre **PRN:**, et **AUX:** pour le port série.

Voir aussi : LIST

P=<processeur>

Autorise le **processeur** ou le **coprocesseur** spécifié parmi 68000, 68010, 68020, 68030, 68EC030, 68040, 68EC040, CPU32, 68881, 68882, 68851. Le choix d'un processeur réinitialise les coprocesseurs.

Voir aussi : MCxxxxx

Défaut : MC68000.

S+/-

Autorise ou interdit la génération de la **table des symboles** en fin de listing.

Défaut : S-

V+/-

Entraîne l'affichage des **statistiques** en fin d'assemblage.

Défaut : V-

W+/-

Autorise ou interdit l'affichage des **avertissements**.

Défaut : W-

X+/-

Demande la génération des symboles **DRI** avec des noms tronqués à 22 caractères au lieu de 8.

Défaut : X-

Y+/-

Demande la génération du **débogage source** (compatible **Pure C**).

Défaut : Y-

FOPT

Syntaxe : **FOPT** <opt>[,<opt>...]

<opt> peut être :

Nom	Explication
ID=<#>	Où # est le numéro d'Id du FPU (par défaut et généralement, 1) (voir MC68881).
K=<#>	Où # est le k-factor utilisé par la conversion interne en packed (17 par défaut).
PREC=	Correspond au PREC du FPCR utilisé lors des calculs flottants. X extended precision (défaut) S single precision D double precision
ROUND=	Correspond au RND du FPCR utilisé lors des calculs flottants. N arrondit au plus proche (défaut) Z arrondit vers zéro N arrondit vers + l'infini M arrondit vers - l'infini

INCLUDE

Syntaxe : **INCLUDE** "<nomdefichier>"

ou **INCLUDE** '<nomdefichier>'

Sert à inclure un fichier source assembleur au source principal en cours d'assemblage comme s'il en faisait partie. Le nombre d'INCLUDE, imbriqués ou non, n'est limité que par la mémoire. En cas d'erreur quelconque (fichier, mémoire, etc ...), ASM affichera une erreur fatale et terminera là l'assemblage.

☞ Le nom du fichier peut ne pas être encadré de guillemets ou d'apostrophes. Nous déconseillons toutefois cette notation par homogénéité avec l'expression générale des chaînes de caractères. Si le nom de fichier ne spécifie pas de chemin absolu, le chemin pris sera relatif à celui du source principal.

INCLUDE	"equates.s"
INCLUDE	"..\COMMON\GLOBALS.ALL"
INCLUDE	"d:\include.s\macros.s"

INCDIR

Syntaxe : INCDIR "<nom de répertoire>"
 ou INCDIR '<nom de répertoire>'
 Ajoute un répertoire dans une liste qui sera parcourue à chaque directive INCLUDE.



INCDIR "d:\includes"
 INCDIR 'f:\include\asm.inc'
 INCDIR f:\include\asm.inc

INCBIN

Syntaxe : INCBIN "<nomdefichier>[,taille[,offset]]"
 ou INCBIN '<nomdefichier>'[,taille[,offset]]'

Synonyme : **IBYTES**

Demande à ASM de lire un fichier quelconque à l'endroit courant d'assemblage et de l'ajouter tel quel dans le fichier généré. Il est possible de préciser la taille à lire (en octets), ainsi que l'offset à partir duquel il faut lire (en octets). Si l'on ne veut que préciser l'offset et pas la taille (pour pouvoir lire le reste du fichier à partir de l'offset précisé), il suffit d'indiquer -1 comme taille. En cas d'erreur quelconque (fichier, mémoire, etc ...), ASM affichera une erreur fatale et terminera là l'assemblage.

Le nom du fichier peut ne pas être encadré de guillemets ou d'apostrophes. Nous déconseillons toutefois cette notation par homogénéité avec l'expression générale des chaînes de caractères. Si le nom de fichier ne spécifie pas de chemin absolu, le chemin pris sera relatif à celui du source principal.



INCBIN "table.bin"
 INCBIN "..\COMMON\KEYS.ALL"
 INCBIN "d:\incbin\hexatab.bin"
 INCBIN "images.lib",32000,64000
 INCBIN "palettes.all",-1,16*3

OUTPUT

Syntaxe : OUTPUT "<nomdefichier>"
 ou OUTPUT '<nomdefichier>'

Spécifie le nom du fichier généré par ASM. Le nom de fichier peut être spécifié intégralement (nom + extension), ou bien sans l'extension, (l'extension par défaut lui sera alors ajoutée), ou bien l'extension seule peut être indiquée, et le nom du source principal (sans l'extension) lui sera ajouté.

Extensions par défaut :

- format exécutable : **PRG**
- format objet Pure C : **O**
- format objet DRI : **O**

Voir aussi : O=<nom> dans la ligne de commande
O=<nom> avec la directive OPT

☞ Le nom du fichier peut ne pas être encadré de guillemets ou d'apostrophes. Nous déconseillons toutefois cette notation par homogénéité avec l'expression générale des chaînes de caractères. Si le nom de fichier ne spécifie pas de chemin absolu, le chemin pris sera relatif à celui du source principal.

 OUTPUT "myprog.prg" ;-> "myprog.prg"
OUTPUT ".prg" ;-> "source.prg"
OUTPUT "myprog" ;-> "myprog.prg"

276

Syntaxe :	EVEN
ou	EVEN.W
ou	EVEN.L
ou	EVEN.Q

Synonyme : **ALIGN**

Sert à forcer la parité du compteur d'assemblage courant, afin d'être aligné à 16 ou 32 bits. EVEN est surtout utile devant ou après les directives DC.B, DS.B, ASCII, ASCIIL et ASCIIZ. En effet, ASM aligne automatiquement à 16 bits les instructions. Si la taille n'est pas indiquée, le mode par défaut est Word (16 bits).

Voir aussi : CNOP

→ L'alignement est toujours relatif au début de la section courante.

 flag: DC.B 1
ptr: EVEN ;assure que ptr sera pair
DC.B "ABCD"

卷之三

Syntaxe : CNOP <offset>. <alignement>

Sert à forcer le compteur d'assemblage courant à l'alignement donné (en octets) et rajoute l'offset (en octets) indiqué.

Voir aussi : EVEN

→ L'alignement est toujours relatif au début de la section courante. La directive EVEN est généralement utilisée à la place de CNOP, bien qu'elle ne permette pas d'indiquer un offset (habituellement non utilisé).

	CNOP	0,2	<code>:= EVEN</code>
	CNOP	0,4	<code>:= EVEN.L</code>
	CNOP	1,2	<code>;aligne à impair le</code> <code>;compteur au prochain</code> <code>;octet</code>

ORG

Syntaxe : ORG <adresse>

Synonyme : RORG

Demande à ASM de générer le code en absolu à partir de l'adresse spécifiée. L'assemblage absolu continue jusqu'à la fin ou jusqu'à une des directives ORG, TEXT, DATA ou BSS.

 ORG 0 ;démarre l'assemblage à 0

SECTION

Syntaxe : SECTION <nom>

Indique à ASM de passer dans la section demandée. <nom> doit être TEXT, DATA ou BSS exclusivement.

Voir aussi : TEXT, DATA, BSS

 SECTION TEXT

SECTION DATA

SECTION BSS

TEXT

Syntaxe : TEXT

Synonyme : CODE

Indique à ASM de passer en section TEXT.

Voir aussi : SECTION, DATA, BSS

 TEXT ;passe en section TEXT

DATA

Syntaxe : DATA

Indique à ASM de passer en section DATA.

Voir aussi : SECTION, TEXT, BSS

 DATA ;passe en section DATA

BSS

Syntaxe : BSS

Indique à ASM de passer en section BSS.

Voir aussi : SECTION, TEXT, DATA

 BSS ;passe en section BSS

OFFSET

Syntaxe : OFFSET [<offset>]

Demande à ASM d'assembler jusqu'à la prochaine déclaration de section ou la fin du source les labels en absolu à partir de l'offset indiqué. Si ce dernier est omis, l'assemblage reprend à partir de la fin du dernier OFFSET.

 OFFSET 0 ;démarre à l'offset 0

END**Syntaxe :** END

Indique à ASM la fin logique du source en cours d'assemblage. Rien ne sera assemblé dans le fichier courant au-delà de cette directive.

N'est pas obligatoire en fin de source.

END ;arrête ici l'assemblage du fichier

FAIL**Syntaxe :** FAIL ["<message>"]

Demande à ASM de générer l'erreur "User Error". Ce n'est pas une erreur fatale, donc l'assemblage continue. Surtout utile en assemblage conditionnel. Cette directive prend un message optionnel à afficher en plus du message "User Error".

IFNE min>max
FAIL ;génère l'erreur "User error"
FAIL "Ptr" ;donne "User error : Ptr"
ENDIF

ERROR**Syntaxe :** ERROR <n°>

Demande à ASM de générer l'erreur <n°>. Ce n'est jamais une erreur fatale, donc l'assemblage continue. Surtout utile en assemblage conditionnel.

IFNE min>max
ERROR 10 ;"Opcode not found"
ENDIF

WARNING**Syntaxe :** WARNING <n°>

Demande à ASM de générer l'avertissement <n°>. Surtout utile en assemblage conditionnel.

IFNE min>max
WARNING 1 ;"0(An) turned to (An)"
ENDIF

Déclaration de variables

EQU

Syntaxe : <label> EQU <valeur>

Synonyme : <label> = <valeur>

Sert à nommer des constantes et des labels. Le label ainsi défini est appelé '**equate**' en anglais, et l'usage perdure en français. Un **equate** peut donc être absolu si l'expression servant à le calculer est de type absolu, ou relatif si elle contient une variable, comme spécifié dans le paragraphe sur les expressions. Dans tous les cas, l'expression doit être résolue au moment de l'évaluation. Elle ne peut contenir de référence avant.

Une utilisation marginale mais néanmoins intéressante de l'**equate** consiste à déclarer une variable locale en globale :



label_global:

...

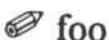
.label_local:

label_local_global EQU .label_local

...

☞ Un **equate** peut être déclaré à tout endroit du programme, y compris après son utilisation (forward reference). Toutefois, pour des raisons d'efficacité, il est conseillé de n'employer cette possibilité que lorsque c'est indispensable.

Voir aussi : SET, FEQU, FSET



foo EQU 3

bar EQU label

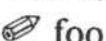
SET

Syntaxe : <label> SET <valeur>

Cette directive est proche de EQU, décrite ci-dessus. Elle sert à définir des variables plutôt que des constantes, car <label> peut se voir assigner une valeur plusieurs fois.

☞ Un **set** doit être déclaré avant sa première utilisation, car sa valeur peut changer et dépend donc de l'endroit où il est utilisé. Il ne peut donc y avoir de "référence avant".

Voir aussi : EQU, FEQU, FSET



foo SET 3

foo SET foo+1

EQU

Syntaxe : <label> EQU <registre>

Sert à nommer un registre pour une utilisation ultérieure dans un mode d'adressage.

Les registres acceptés sont D0 à D7, A0 à A7, SP, PC, ZA0 à ZA7, ZPC ou un autre EQU.

Voir aussi : REG, FEQUR

☞ compteur EQU d5

pointeur EQU a2

pointeur EQU pc

REG

Syntaxe : <label> REG <liste de registres>

Sert à nommer une liste de registres pour une utilisation ultérieure dans une instruction MOVEM.

Les registres acceptés sont, comme pour l'instruction MOVEM :

D0 à D7, A0 à A7, SP ainsi que les EQU (de D0 à A7) ou un autre **reg**.

☞ Un **reg** doit être défini avant son utilisation.

La syntaxe de la liste de registres suit exactement la syntaxe définie dans l'instruction MOVEM, ainsi que les messages d'erreur et d'avertissement afférents.

Voir aussi : MOVEM, EQU, FREG

☞ reglist REG d0-d2/d4/d6-d7/a2-a6

movem.l reglist,-(sp)

movem.l (sp)+,reglist

pointeur EQU a2

reglist REG d0-d2/d4/d6-d7/pointeur-a6

Déclaration de variables pour virgule flottante

FEQU

Syntaxe : <label> FEQU <valeur>

Sert à nommer des constantes flottantes. L'expression doit être résolue au moment de l'évaluation. Elle ne peut contenir de référence avant.

☞ A la différence des équates "normaux" (déclarés avec EQU), les équates flottantes ne peuvent être déclarées après leur utilisation.

Voir aussi : EQU, FSET

☞ deux FEQU \$40000000

pi FEQU 3.0+0.1416

pi FEQU fpurom(0)

FSET

Syntaxe : <label> FSET <valeur>

Sert à nommer des variables flottantes. L'expression doit être résolue au moment de l'évaluation. Elle ne peut contenir de référence avant.

☞ Comme les **sets** "normaux" (déclarés avec SET), les **sets** flottants ne peuvent être déclarés après leur utilisation.

Voir aussi : SET, FEQU

☞ fact FSET 0.5

fact FSET 1.0+sqrt(fact)

☞ Il n'est pas nécessaire de préciser la taille d'un FEQU ou d'un FSET. Leur valeur est toujours stockée en Extended.

FREG

Syntaxe : <label> FREG <liste de registres>

Sert à nommer une liste de registres pour une utilisation ultérieure dans une instruction FMOVE.M.

Les registres acceptés sont, comme pour l'instruction FMOVE.M :

FP0 à FP7, FPCR, FPSR, FPIAR, D0 à D7 ainsi que les FEQUR et les EQUR (de D0 à D7) (ou un autre FREG).

☞ Un FREG doit être défini avant son utilisation. La syntaxe de la liste de registres suit exactement la syntaxe définie dans l'instruction FMOVE.M.

Voir aussi : FMOVE.M, REG

reglist FREG fp0-fp2/fp4/fp6-fp7
fmove.m reglist,-(sp)
fmove.m (sp)+,reglist

compte FEQUR fp2
reglist FREG fp0-compte/fp4/fp6-fp7

FEQUR

Syntaxe : <label> FEQUR <registre>

Sert à nommer un registre FPU pour une utilisation ultérieure dans un mode d'adressage.

Les registres acceptés sont FP0 à FP7 ou un autre FREG.

Voir aussi : EQUR

coef1 FEQUR fp1 :coef1 signifie fp1
mul FEQUR fp7 :mul signifie fp7

Déclaration de données initialisées

DC

Syntaxe : DC <valeur>[,valeur][,valeur]...
 ou DC.B <valeur>[,valeur][,valeur]...
 ou DC.W <valeur>[,valeur][,valeur]...
 ou DC.L <valeur>[,valeur][,valeur]...

Voir aussi : DCB, ASCII

Signifie **Define Constant**. Permet la déclaration de données prédéfinies de la taille indiquée. Il est possible de déclarer une suite de valeurs séparées par une virgule, à concurrence de la taille maximum d'une ligne (256 caractères), soit au plus 127.

DC comprend une syntaxe supplémentaire destinée à la déclaration de chaînes de caractères. Il suffit d'écrire la suite de caractères encadrée par des guillemets.

 DC 1,2,4 ;génère les mots de 2 octets 1, 2, 4
 DC.B 0,1 ;génère les octets 0 et 1
 DC.B "ASCII",0 ;génère la suite d'octets ASCII
 DC.W 1,2,4,8 ;génère les mots de 2 octets
 1,2,4,8
 DC.L 4 ;génère le long 4

DCB

Syntaxe : DCB <nombre>[,valeur]
 ou DCB.B <nombre>[,valeur]
 ou DCB.W <nombre>[,valeur]
 ou DCB.L <nombre>[,valeur]

Voir aussi : DC, DS

Signifie **Define Constant Block**. Permet de déclarer un bloc de données en définissant le nombre d'éléments et la valeur de l'élément. Si cette dernière est omise, 0 est pris par défaut.

 DCB 256,0 ;génère 256 mots à 0
 DCB.B 128,-1 ;génère 128 octets à -1
 DCB.W 64 ;équivalent à DS.W 64
 DCB.L 256,\$f ;génère 256 longs à \$f

ASCII

Syntaxe : ASCII <"chaîne">

ou : ASCII <'chaîne'>

Voir aussi : ASCIIZ, ASCIIL, DC

Sert à déclarer une chaîne de caractères, de la même manière que DC.B.

 ASCII "Chaîne en ASCII"

ASCIIZ

Syntaxe : ASCIIZ <"chaîne">

ou : ASCIIZ <'chaîne'>

Sert à déclarer une chaîne de caractères, de la même manière que DC.B, et y ajoute automatiquement un octet à 0.

Voir aussi : ASCII, ASCIIL, DC

 ASCIIZ "Chaîne en ASCII terminée par 0"

ASCIIL

Syntaxe : ASCIIL <"chaîne">

ou : ASCIIL <'chaîne'>

Sert à déclarer une chaîne de caractères, de la même manière que DC.B, et insère automatiquement à son début un octet contenant la longueur de la chaîne (**chaîne Pascal**).

Voir aussi : ASCII, ASCIIZ, DC

 ASCIIL "Chaîne précédée de sa longueur"

Déclaration de données non initialisées

DS

Syntaxe : DS <nombre>
 ou DS.B <nombre>
 ou DS.W <nombre>
 ou DS.L <nombre>

Voir aussi : DCB

Signifie **Define Storage**. Permet de résERVER de l'espace en section BSS ou OFFSET. La taille réservée est calculée en multipliant la valeur indiquée par la taille de l'instruction.

☞ Si cette directive est utilisée ailleurs qu'en section BSS ou OFFSET, une quantité équivalente d'octets de valeur 0 sera générée dans le code.

	DS 3	:réserve 3*2=6 octets
	DS.B 50	:réserve 50*1=50 octets
	DS.W 10	:réserve 10*2=20 octets
	DS.L 1	:réserve 1*4=4 octets

Section RS

Introduction

Les directives RS se rapprochent des STRUCTures du C ou des RECORD du Pascal. RS sert à déclarer des constantes sans avoir à leur donner de valeur explicite. A chaque déclaration, le compteur interne de RS (variable _RS) est incrémenté de la taille spécifiée. Les autres directives RS servent à manipuler ce compteur.

RS

Syntaxe : Label RS <nombre>
 ou RS.B <nombre>
 ou RS.W <nombre>
 ou RS.L <nombre>

Voir aussi : DS

Permet de déclarer un label dont la valeur (absolue) est égale à celle du compteur _RS. Ce dernier est ensuite incrémenté du nombre spécifié multiplié par la taille de l'instruction.

	Label RS 3	;additionne 3*2=6 octets
Label	RS.B 50	;additionne 50*1=50 octets
Label	RS.W 10	;additionne 10*2=20 octets
Label	RS.L 1	;additionne 1*4=4 octets

RSSET

Syntaxe : RSSET <valeur>

Voir aussi : RS

Change la valeur du compteur __RS avec la valeur spécifiée.

RSSET 4 ;__RS=4

RSRESET

Syntaxe : RSRESET

Voir aussi : RS

Remet la valeur du compteur __RS à 0.

RSRESET ;__RS=0

RSEVEN

Syntaxe : RSEVEN

ou RSEVEN.W

ou RSEVEN.L

ou RSEVEN.Q

Voir aussi : RS

Sert à forcer la parité du compteur __RS, afin d'être aligné à 16, 32 ou 64 bits. Si la taille n'est pas indiquée, le mode par défaut est Word (16 bits).

RSEVEN ;force __RS à pair
RSEVEN.W ;force __RS à pair
RSEVEN.L ;force __RS au long

RSSTRUCT

Syntaxe : RSSTRUCT

Voir aussi : RSUNION

RSEND

Sert à déclarer des STRUCTures (équivalent au STRUCT du C ou RECORD du Pascal). Le nom de la structure est obligatoire. Chaque RS suivant sera considéré comme faisant partie de cette structure, jusqu'au RSEND suivant. Ainsi, le label qui sera effectivement créé sera <nom de structure>.<label>. Au moment du RSEND, le label <nom de structure>.sizeof est créé prenant comme valeur la taille de la structure.

image rsstruct
x rs.l 1
y rs.l 1
planes rs.w 1
comp rs.w 1
rsend

génère en réalité les labels suivants :

image.x valant 0

image.y	valant	4
image.planes	valant	8
image.comp	valant	10
image.sizeof	valant	12.

ce qui permet de les utiliser comme des équates normaux :

☞

```

moveq #image.sizeof,d0
bsr malloc
beq error
move.l d0,Image
...
move.l Image,a0
move.l image.x(a0),d0
move.l image.y(a0),d1
move.w image.planes(a0),d2
bsr display_image
...

```

RSUNION

Syntaxe : RSUNION
 Voir aussi : RSSTRUCT
 RSEND

Sert à déclarer des UNIONS (équivalent au UNIONS du C). Le nom de l'union est obligatoire. Chaque RS suivant sera considéré comme faisant partie de cette union, jusqu'au RSEND suivant. Ainsi, le label qui sera effectivement créé sera <nom d'union>.<label>. Au moment du RSEND, le label <nom d'union>.sizeof est créé prenant comme valeur la taille de l'union.

☞ ptr rsunion
 bytes rs.b 4
 long rs.l 1
 rsend

génère en réalité les labels suivants :

ptr.bytes valant 0
 ptr.long valant 0

ce qui permet de les utiliser comme des équates normaux.

RSEND

Syntaxe : RSEND
 Voir aussi : RSSTRUCT
 RSUNION

Sert à terminer une déclaration de structure ou d'union.

Déclaration de paramètres

CARGS

Syntaxe : CARGS [#<offset>],<label.sz>...

Voir aussi : PARGS

Sert à définir le passage de paramètres au format **C** par la pile. Les labels sont créés de gauche à droite, leur valeur part d'offset et est incrémentée de la taille spécifiée. Si offset est omis, il prend pour valeur 4.

☞ La taille spécifiée peut être B (1), W (2) ou L (4). Cependant, quand le processeur empile un octet, il décrémente la pile de 2 octets pour s'assurer qu'elle reste toujours paire. Ainsi, si une fonction C passe un octet sur la pile à une autre fonction, en réalité, l'octet est passé comme un mot (16 bits). Donc, pour récupérer un paramètre à l'octet (char), il faut le reprendre en Word, ou en Byte en ajoutant 1 au label.

```

é ;en C : foo (short par1 , long par2);
foo:    CARGS par1.w.par2.l
          move.w  par1(sp),d0      ;donne 4(sp)
          move.l   par2(sp),d1      ;donne 6(sp)

é ;en C : bar (char par1 , short par2)
bar:    CARGS par1.w.par2.w
          move.w  par1(sp),d0      ;équivalent pour
          move.b   par1+1(sp),d0    ;récupérer par1 en
          move.w   par2(sp),d1      ;Byte (dans D0.B)

```

PARGS

est identique à CARGS, mais crée les labels de la droite vers la gauche pour une routine appelée en **Pascal**.

```

é ;en pascal : foo(int par1 , long par2);
foo:    PARGS par1.w.par2.l
          move.w  par1(sp),d0      ;donne 8(sp)
          move.l   par2(sp),d1      ;donne 6(sp)

```

Contrôle du listing

LIST

Syntaxe : LIST
ou LIST +
ou LIST -

Sans opérande, cette directive déclenche la sortie du **listing** d'assemblage. Avec l'opérande +, le compteur interne de listing sera incrémenté de 1, et avec -, il sera décrémenté de 1. Si ce compteur est supérieur ou égal à 0, alors le listing est en marche. Par défaut, la valeur de ce compteur est -1 (pas de listing). La directive LIST passe ce compteur à 0, et NOLIST le passe à -1.

Voir aussi : NOLIST

LIST ;:listing en route
LIST + ;:idem
LIST - ;:arrêt si arrêté avant LIST+

NOLIST

Syntaxe : NOLIST

Arrête immédiatement toute sortie de listing (passe le compteur interne de listing à -1).

Voir aussi : LIST

NOLIST ;:arrête la sortie du listing

SPC

Syntaxe : SPC <nombre>

Ajoute au listing <nombre> lignes vides.

SPC 10 ;:saute 10 lignes

TTL

Syntaxe : TTL <"titre">

ou TTL <'titre'>

Le titre apparaîtra centré sur la première ligne de chaque page de listing.

TTL "Titre principal"

SUBTTL

Syntaxe : SUBTTL <"titre">

ou SUBTTL <'titre'>

Le sous-titre apparaîtra centré sur la deuxième ligne de chaque page de listing, en dessous du titre.

SUBTTL "Titre secondaire"

PAGE

Syntaxe : **PAGE**

ou **PAGE <nombre>**

Si PAGE est utilisé sans paramètre, un saut de page est généré dans le listing. Si un paramètre est donné, il symbolise la longueur d'une page en nombre de lignes de caractères.

Par défaut, une page contient 60 lignes.

☞ Si le listing n'est pas activé, cette directive (dans la forme sans paramètre) n'a aucun effet.

 **PAGE** ;génère un saut de page
PAGE 58 ;fixe la taille à 58 lignes

NOPAGE

Syntaxe : **NOPAGE**

Arrête la sortie de pages formatées (titres, sauts de page, ...).

 **NOPAGE**

LLEN

Syntaxe : **LLEN <nombre>**

Règle la longueur d'une ligne en nombre de caractères. Par défaut, une ligne contient 132 caractères.

 **LLEN 132** ;longueur : 132 caractères

PLEN

Syntaxe : **PLEN <nombre>**

Règle la longueur d'une page en nombre de lignes. Par défaut, une page contient 66 lignes.

 **PLEN 66** ;longueur : 66 lignes

Voir aussi : **PAGE <#>**

LISTCHAR

Syntaxe : **LISTCHAR <code>[,<code>]**

Sert à envoyer des **codes de contrôle** dans le listing.

 **LISTCHAR \$c** ;envoie un saut de page

TABS

Syntaxe : **TABS [<nombre>]**

Permet de demander l'expansion des **tabulations** en espace en précisant éventuellement le nombre d'espaces pour une tabulation. Par défaut, une tabulation vaut 8 espaces.

 **TABS** ;1 tab = 8 espaces
TABS 4 ;1 tab = 4 espaces

Directives externes

XREF

Syntaxe : XREF <label>[,<label>][...]

Synonyme : **IMPORT** <label>[,<label>][...]

Indique à ASM qu'un label n'est pas défini dans le source en cours d'assemblage, mais doit être retrouvé au moment de l'édition de liens.

Voir aussi : XDEF

 XREF strcpy,strcmp

XDEF

Syntaxe : XDEF <label>[,<label>][...]

Synonyme : **EXPORT** <label>[,<label>][...]

Indique que le label doit être accessible à partir d'un autre fichier objet.

Voir aussi : XREF

 XDEF copyblock



Directives diverses

SUPER

Syntaxe : SUPER

Autorise l'utilisation des instructions ne fonctionnant qu'en mode superviseur (mode par défaut).

Voir aussi : OPT u-

USER

 SUPER

USER

Syntaxe : USER

Interdit l'utilisation des instructions ne fonctionnant qu'en mode superviseur.

 USER

Voir aussi : OPT u+

SUPER

MC68000

Syntaxe : MC68000

N'autorise l'assemblage que des instructions 68000 (et non au-delà). A utiliser notamment en cas de **68EC000**, **68HC000**, **68008** et **68300**.

 MC68000

Voir aussi : OPT p=68000

MC68010

Syntaxe : MC68010

Autorise l'assemblage des instructions **68010** : BKPT, MOVE from CCR, MOVEC, MOVES, RTD.

 MC68010

Voir aussi : OPT p=68010

MC68020

Syntaxe : MC68020

Autorise l'assemblage des instructions **68010+68020** : BFxxx, CALLM, CAS, CAS2, CHK2, CMP2, cpxxx, DIVSL, DIVUL, EXTB, PACK, RTM, TRAPcc, UNPACK.

 MC68020

Voir aussi : OPT p=68020

MC68030

Syntaxe : MC68030

Autorise l'assemblage des instructions 68010+**68030** : BFxxx, CAS, CAS2, CHK2, CMP2, cpxxx, DIVSL, DIVUL, EXTB, PACK, PFLUSH, PFLUSHA, PLOAD, PMOVE, PTEST, TRAPcc, UNPACK.

 MC68030

Voir aussi : OPT p=68030

MC68EC030

Syntaxe : MC68EC030

Autorise l'assemblage des instructions 68030-MMU : BFxxx, CAS, CAS2, CHK2, CMP2, cpxxx, DIVSL, DIVUL, EXTB, PACK, PMOVE, PTEST, TRAPcc, UNPACK.

 MC68EC030

Voir aussi : OPT p=68EC030

MC68040

Syntaxe : MC68040

Autorise l'assemblage des instructions 68010+**68040** : BFxxx, CAS, CAS2, CHK2, CINV, CMP2, cpxxx, CPUSH, DIVSL, DIVUL, EXTB, Fxxx, MOVE16, PACK, PFLUSH, PTEST, TRAPcc, UNPACK. Le 68LC040 s'obtient en déclarant :

MC68040

MC68881

 MC68040

Voir aussi : OPT p=68040

MC68EC040

Syntaxe : MC68EC040

Autorise l'assemblage des instructions 68040-MMU-FPU : BFxxx, CAS, CAS2, CHK2, CINV, CMP2, cpxxx, CPUSH, DIVSL, DIVUL, EXTB, Fxxx, MOVE16, PACK, PTEST, TRAPcc, UNPACK.

 MC68EC040

Voir aussi : OPT p=68EC040

MCCPU32

Syntaxe : MCCPU32

Autorise l'assemblage des instructions **CPU32**. L'assemblage du mode d'adressage (Bd, Pn, XnSzSc) est aussi autorisé. A utiliser notamment en cas de **68330** et **68340**.

 MCCPU32

Voir aussi : OPT p=CPU32

MC68881

Syntaxe : MC68881 [id] ou [-]

Autorise l'assemblage des instructions **68881** ou **68882**. Le numéro de coprocesseur peut éventuellement être spécifié. Un moins (-) en interdit l'assemblage.

 MC68881 7 :autorise un FPU d'id 7

Voir aussi : OPT p=68881
FOPT id=

MC68851

Syntaxe : MC68851 [-]

Autorise l'assemblage des instructions **68851**. Un moins (-) en interdit l'assemblage.

 MC68851

Voir aussi : OPT p=68851

Macros

Introduction

Ce chapitre traite de la partie **macro-assembleur** d'ASM. Une **macro** sert à définir une séquence d'instructions et de directives à laquelle on peut passer des paramètres et fixer une taille, à chaque appel de la macro dans le corps du programme.

Vue générale

Les macros possèdent 2 phases distinctes, la déclaration et l'expansion.

Une **déclaration de macro** commence par son nom. Celui-ci servira à son appel ultérieur, en l'écrivant dans le champ opcode. Il est interdit d'utiliser le nom d'un mnémonique ou d'une directive.

Toute écriture valide en général peut être mise en macro, sauf la déclaration de macro.

L'**expansion de macro** se produit au moment de son appel. Son contenu est analysé et les paramètres substitués. Puis le nouveau corps vient s'insérer à l'endroit courant du source, et est assemblé normalement.

L'appel s'écrit simplement en écrivant le nom de la macro dans le champ opcode, et les paramètres éventuels dans les champs opérandes.

Définir une macro

Une définition de macro contient 3 parties :

- l'**en-tête de la macro**, contenant le nom de la macro suivi de la directive MACRO.
- le **corps de la macro**, contenant une suite de lignes de source contenant éventuellement des arguments.
- la **fin de macro** indiquée par la directive ENDM.

En-tête

MACRO

Syntaxe : <label> MACRO

Le champ <label> servira de nom à la macro, par lequel elle pourra être appelée. Ce label ne peut être un nom d'instruction ou de directive.

Corps

Il contient une suite de lignes de source contenant éventuellement des arguments.

☞ Si un commentaire est signalé par le doublement du symbole (;; ou **), alors il ne sera pas mémorisé dans le corps de la macro.

SHIFTM

Syntaxe : **SHIFTM**

Sert à décaler les paramètres de la macro. Cela permet de parcourir une liste de paramètres en les utilisant un par un.

```
✍ enum          MACRO
\2             SET    \1
                IF   \#>=3
                  REPT  \#-2
                    SHIFTM
                    SET \1+1
\2
                ENDR
              ENDIF
            ENDM
```

Fin de déclaration**ENDM**

Syntaxe : **ENDM**

Annonce la fin de déclaration et d'expansion de macro.

Fin d'expansion**EXITM**

Syntaxe : **EXITM**

Synonyme : **MEXIT**

Termine l'expansion de la macro. L'assemblage reprendra après le ENDM suivant. Surtout utile en assemblage conditionnel.

```
✍ foo          MACRO
              IF  \1>2
                EXITM
              ELSEIF
                foo \1+2
              ENDIF
            ENDM
```

Paramètres

Des paramètres peuvent être transmis à la macro, numérotés de **\1 à \9** puis de **\A à \Z**. Les noms de paramètre sont remplacés par les arguments tels quels. Pour écrire un vrai **** dans le corps de la macro, il est nécessaire de le doubler (****). Le paramètre **\0** sert à récupérer la taille déclarée à l'appel de macro (par défaut W), et **\#** contient le nombre de paramètres.

Si un paramètre transmis contient une virgule ou un espace blanc (non encadré par des guillemets ou des apostrophes), il doit être encadré de signes supérieur et inférieur (**<>**). De ce fait, pour pouvoir passer un signe inférieur (**>**) tel quel dans un paramètre, il est nécessaire de l'échapper en le doublant (**>>**).

Appel

On appelle une macro par :

<nom>[.<taille>] [<paramètre>]

A l'appel d'une macro et si elle contient elle-même un appel de macro, cette dernière doit avoir été définie au préalable. Elle n'a pas besoin d'avoir été déclarée au moment de la déclaration de la macro appelante.

Si un appel de macro doit dépasser une ligne de source (256 caractères maximum), il est possible de continuer sur la ligne suivante en mettant comme premier caractère de celle-ci un "et commercial" (**&**). La ligne doit être coupée entre deux paramètres, après la virgule.

Après la reconnaissance de la directive MACRO, le nom de la macro est cherché dans les mnémoniques et directives. S'il correspond à l'un d'eux, une erreur est alors générée.

 Ceci est un exemple de macro de recopie de chaîne de caractères terminée par 0.

```
copy MACRO
.\@: move.b (\1)+,(\2) +
      bne.s .\@
      ENDM
```

Cette macro s'attend à trouver comme premier paramètre un registre d'adresse pointant sur la chaîne source, et comme deuxième paramètre un registre d'adresse pointant sur la chaîne destination.

L'adresse du branchement possède une syntaxe étrange, destinée à générer un label unique à chaque nouvel appel de macro. Le \@ est remplacé par un label de la forme _xxx, où x est un chiffre décimal (0 à 9). xxx est la valeur d'un compteur interne, démarrant à 001 et incrémenté de 1 à chaque macro utilisant \@. Si le nombre d'utilisation de \@ dépasse 999, le nombre est généré tel quel (ex : 1001 donne _1001).

Le nombre de chiffres fixe peut être changé avec (OPT) @<#> où # est le nombre (de 1 à 9) de chiffres minimum.

Voir aussi : OPT @#

Expansion du corps

Continuons notre exemple avec l'appel suivant :

copy a0,a1

ASM va d'abord chercher si copy est un mnémonique ou une directive. Si tel n'est pas le cas, la table de macros sera vérifiée. Si la recherche est infructueuse, ASM préviendra que l'instruction n'a pas été trouvée.

Voici le résultat de l'expansion de la macro :

_001: move.b (a0)+,(a1)+
bne.s .001

Le paramètre \1 a été remplacé par a0, \2 par a1, et \@ par _001.

Les appels récursifs de macro ne sont limités que par la mémoire (Ils comportent d'ailleurs généralement la directive EXITM).

Le nombre d'arguments peut être connu par \@ ou par la variable interne _NARG (ce dernier usage est cependant déconseillé).

Un symbole peut être remplacé par sa valeur décimale ou hexadécimale en écrivant \<nomdesymbole> ou \<\$nomdesymbole>. Le symbole doit bien sûr avoir été déjà défini et ne pas être relatif.

Une utilisation possible est l'affichage à l'écran de la valeur d'un symbole durant l'assemblage.

opt M+
printval MACRO
xval SET \1
LIST +
:la valeur de \1 est \<xval>
LIST -
ENDM

Un numéro de paramètre peut aussi être obtenu dynamiquement. La syntaxe `<nomdesymbole> signifie: paramètre numéro <valeur de nomdesymbole>. On peut aussi connaître la taille d'un paramètre avec \?<n°deparamètre>.



```

init      MACRO  ;:init de données en ordre inverse
nb       set \#   ;:nombre de paramètres
        REPT   \#
        IF \?<nb>  ;:ce paramètre existe-t-il?
        dc.\0  \`<nb>  ;:oui alors init
        ENDIF
nb       set nb-1    ;:paramètre suivant
        ENDR
        ENDM

```

☞ L'exemple le plus classique d'utilisation de macro est l'appel de librairies :

```

pstring   MACRO
          lea      \1,a0
          bsr      Cconws
          ENDM
string:   dc.b    "Bla bla bla"
          even
          pstring string

```

Ce peut être un calcul mathématique récursif :

```

fac      MACRO  ;:fac(variable,n)
        IFND   \1  ;:la variable existe-t-elle déjà?
        set 1   ;:non alors init à 1 (exp(0))
        ENDIF
        IFGT   \2  ;:n>2
        fac \1,\2-1 ;:fac(n-1)
        set \1*(\2) ;:n=n*fac(n-1)
        ENDIF
        ENDM
        fac foo,3
donnera : foo=3*(3-1)=6.

```

L'encadrement du deuxième paramètre par des parenthèses permet d'être tranquille en cas de macro expansion d'une expression complexe. Il ne pourra y avoir d'éventuels conflits de priorité d'opérateurs.

On peut utiliser une macro pour la maintenance du programme :

```

INFO      MACRO
        dc.b    \1,"\<$version>",0
        ENDM
version   EQU     $101
INFO      "Version du programme: "
donnera:  dc.b    "Version du programme: ","101",0

```

Récapitulatif des codes spéciaux en macro:

Code	Description
\`	Génère un \
\@	Génère un label unique
\#	Contient le nombre d'arguments de la macro
\0	Contient la taille passée à la macro
\1 à \9	Les 9 premiers paramètres
\A à \Z	Les 26 derniers paramètres
\?<par>	Contient la taille du paramètre par
`<label>	Génère la valeur de label comme n° de paramètre
<label>	Génère la valeur de label en décimal
<\$label>	Génère la valeur de label en hexadécimal
&	Ignore la fin de ligne

Assemblage conditionnel

Introduction

L'**assemblage conditionnel** sert à pouvoir générer des exécutables différents à partir d'un même source, ou à vérifier que l'assemblage se passe correctement.

Il se décompose en 3 parties :

- un en-tête avec une expression conditionnelle (IF, IFC, ...) et une succession de lignes de source destinées à être assemblées si la condition est vraie.
- une directive ELSE optionnelle servant à assembler les lignes de source suivantes dans le cas où cette condition serait fausse.
- une directive ENDIF signifiant la fin de l'expression conditionnelle.

En-tête

Il existe deux formes de comparaison arithmétique. L'une sert à comparer deux chaînes de caractères (IFC et IFNC). L'autre teste des nombres (compare par rapport à 0) : les IFcc, cc valant :

cc	Signification	Équivalent
NE	Not Equate	expr!=0
EQ	Equate	expr==0
GT	Greater Than	expr>0
GE	Greater or Equal	expr>=0
LT	Lower Than	expr<0
LE	Lower or Equal	expr<=0

Si l'expression testée vérifie la condition correspondant au test effectué, l'assemblage se poursuit normalement, jusqu'au prochain ELSE ou ENDIF du même niveau. Si ELSE est rencontré, le reste du source est sauté jusqu'au prochain ENDIF de même niveau. Après ce ENDIF, l'assemblage reprend normalement.

Les expressions acceptées suivent la syntaxe standard d'ASM. On peut donc comparer deux adresses, mais la comparaison se fera sur la base du début de la section de chacune.

☞ L'expression doit être parfaitement évaluable au moment de l'assemblage. Les références avant (Forward Reference) sont interdites.

IF

Syntaxe : IF <expression>

Synonyme : IFNE

Voir aussi : IFEQ

Vérifie que <expression> est différente de 0

 IF foo
FAIL "foo est différent de 0!"
ENDIF

IFEQ

Syntaxe : IFEQ <expression>

Voir aussi : IFNE

Vérifie que <expression> est égale à 0

 IFEQ foo
FAIL "foo vaut 0!"
ENDIF

IFGT

Syntaxe : IFGT <expression>

Vérifie que <expression> est strictement supérieure à 0

 IFGT foo
FAIL "foo supérieur à 0!"
ENDIF

IFGE

Syntaxe : IFGE <expression>

Vérifie que <expression> est supérieure ou égale à 0

 IFGE foo
FAIL "foo supérieur ou égal à 0!"
ENDIF

IFLT

Syntaxe : IFLT <expression>

Vérifie que <expression> est strictement inférieure à 0

 IFLT foo
FAIL "foo inférieur à 0!"
ENDIF

IFLE

Syntaxe : IFLE <expression>

Vérifie que <expression> est inférieure ou égale à 0

 IFLE foo
FAIL "foo inférieur ou égal à 0!"
ENDIF

IFC

Syntaxe : IFC <"chaîne1">,<"chaîne2">

Voir aussi : IFNC

Vérifie que les deux chaînes sont semblables (avec différenciation majuscule-minuscule).



```
IFC      "abcd","abcd"
FAIL    "Chaînes identiques!"
ENDIF
```

IFNC

Syntaxe : IFNC <"chaine1">,<"chaine2">

Voir aussi : IFC

Vérifie que les deux chaînes sont dissemblables.



```
IFNC    "abc","abcd"
FAIL   "Chaînes dissemblables!"
ENDIF
```

IFD

Syntaxe : IFD <label>

Voir aussi : IFND

Vérifie que le label est défini (déjà déclaré dans le programme).



```
IFD     foo
FAIL   "foo déjà déclaré"
ENDIF
```

IFND

Syntaxe : IFND <label>

Voir aussi : IFD

Vérifie que le label n'a pas été déjà défini.



```
IFND   foo
FAIL  "foo pas encore déclaré"
ENDIF
```

Alternative**ELSE**

Syntaxe : ELSE

Débute la suite d'instructions qui sera assemblée si la condition testée est fausse.



```
IF      foo
FAIL  "foo différent de 0"
ELSE
FAIL  "foo égal à 0"
ENDIF
```

On dispose aussi des variantes en ELSEIFcc :

**ELSEIF, ELSEIFNE, ELSEIFEQ, ELSEIFGT, ELSEIFGE, ELSEIFLT,
ELSEIFLE, ELSEIFC, ELSEIFNC, ELSEIFD, ELSEIFND.**

Ces dernières permettent de directement tester une nouvelle condition si la première est fausse.

Fin

ENDIF

Syntaxe : **ENDIF**

Synonyme : **ENDC**

Termine le bloc d'assemblage conditionnel. Les blocs peuvent être imbriqués jusqu'à concurrence de la mémoire disponible, mais chaque niveau doit se terminer par un ENDIF.

Assemblage conditionnel en macro

L'exécution des macros est souvent régie par un assemblage conditionnel. Il est par exemple possible de tester la présence d'un paramètre en le comparant avec une chaîne vide :

 IFC "", "\1" ou IFC " \1," "

pour le paramètre 1.

La sortie prématurée d'une macro se fait aussi en assemblage conditionnel :

```
    IF \1<2  
        EXITM  
    ENDIF
```

Dans ce cas, la directive EXITM remplace ENDM et ENDIF du même coup. Dans les autres cas, la structure de macro est prioritaire sur la structure d'assemblage conditionnel. En effet, il est possible de commencer une condition en appelant une macro et de la finir en en appelant une autre.

Assemblage répété

REPT

Syntaxe : REPT <nombre>

Provoque l'assemblage répété <nombre> de fois du bloc d'instructions compris entre REPT et ENDR. On peut imbriquer les blocs REPT/ENDR.

Voir aussi : ENDR



```
REPT    5
REPT    2
nop
ENDR
ENDR
```

::génère 10 nops

ENDR

Syntaxe : ENDR

Termine un bloc de REPT.



```
foo      EQU    3
REPT    foo
nop
ENDR
```

::génère 3 nops

Appendices

Appendice A • Variables réservées

ASM

Contient le numéro de version d'ASM.

DBG

Contient le type de débogage :

0	pas de débogage
1	OPT D
2	OPT X

Voir aussi : OPT d
 OPT x

FPU

Contient le processeur mathématique autorisé :

0	aucun
1	68881-2
2	68040

Voir aussi : OPT p=
 MCxxxxx

LK

Contient le type du fichier généré :

0	exécutable
1	objet Pure C
2	objet DRI

Voir aussi : OPT l<#>

MMU

Contient le processeur de gestion de mémoire autorisé :

0	aucun
1	68851
2	68030
3	68040
4	version EC des 68030/40

Voir aussi : OPT p=xxxxx
 MCxxxxx

MPU

Contient le processeur principal autorisé :

0	68000
1	68010
2	CPU32
3	68020
4	68030
5	68040

Voir aussi : OPT p=xxxxx
 MCxxxxx

RS

Contient la valeur du compteur RS courant.

Voir aussi : RS

NARG

Contient le nombre de paramètres passés à la macro en cours d'exécution (équivalent à \#).

Voir aussi : MACRO, \#

Appendice B • Optimisations

N°	Explication
1	Raccourcit les branchements arrières si le déplacement est compris entre -128 et -1.
2	d16(An) est converti en (An) si d16 vaut 0.
3	Les adresses absolues comprises entre -32768 et +32767 sont converties en mot.
4	Les MOVE.L #i,Dn où i est compris entre -128 et +127 sont convertis en MOVEQ #i,Dn.
5	Les ADDI et SUBI #i sont convertis en ADDQ et SUBQ si i est compris entre 1 et 8.
6	Les branchements avant optimisables en branchement court (0 à +127) sont signalés par un message d'avertissement.
7	Les branchements courts (avant) pointant immédiatement en dessous d'eux-mêmes, ce qui est interdit, sont convertis en NOP.
8	Le base displacement est optimisé de deux manières <ul style="list-style-type: none"> • le mode d'adressage est : (Bd,An,XnSizeScale) ou (bd,PC,XnSizeScale). Si dans ce cas Bd est compris entre -128 et +127, ce mode est converti en d8(An,XnSizeScale) ou d8(PC,XnSizeScale). • sinon et dans tous les autres cas, si Bd est compris entre -32768 et +32767, sa taille devient mot, sauf si Bd vaut 0, auquel cas il est supprimé.
9	L'outer displacement est optimisé en mot si Od est compris entre -32768 et +32767 sauf s'il vaut 0, auquel cas il est supprimé.
10	Transforme les ADDA/SUBA #i,An en LEA i(An),An si i est compris entre -8 et 8 et O5 n'est pas sélectionné, ou si i est compris entre -32768 et +32767.
11	Transforme les LEA d16(An),An en ADDQ/SUBQ si d16 est compris entre 1 et 8.
12	Transforme les MOVEA.L #i,An en MOVEA.W #i,An si i est compris entre -32768 et +32767.
13	Transforme les MOVEA.x #0,An en SUBA.L An,An.
14	Transforme les MOVE.W #0 en CLR.W.
15	Transforme les CMP #0 en TST.
16	Transforme les LSL #1,Dn en ADD Dn,Dn.
17	Transforme les ROXL #1,Dn en ADDX Dn,Dn.

Appendice C • Directives

=	EXPORT	MEXIT
ABS	FAIL	NOFORMAT
ALIGN	FEQU	NOL
ASCII	FEQUR	NOLIST
ASCIIL	FOPT	NOOBJ
ASCIIZ	FREG	NOPAGE
BSS	FSET	OFFSET
CARGS	IBYTES	OPT
CNOP	IF	ORG
CODE	IFC	OUTPUT
DATA	IFD	PAGE
DC	IFEQ	PARGS
DCB	IFGE	PLEN
DS	IFGT	PRINTER
EJECT	IFLE	REG
ELSE	IFLT	REPT
ELSEIF	IFNC	RORG
ELSEIFC	IFND	RS
ELSEIFD	IFNE	RSEND
ELSEIFEQ	IMPORT	RSEVEN
ELSEIFGE	INCBIN	RSRESET
ELSEIFGE	INCDIR	RSSET
ELSEIFGT	INCLUDE	RSSTRUCT
ELSEIFLT	LIST	RSUNION
ELSEIFNC	LISTCHAR	SECTION
ELSEIFND	LLEN	SET
ELSEIFNE	MACRO	SHIFTM
END	MC68000	SPC
ENDC	MC68010	SUBTTL
ENDIF	MC68020	SUPER
ENDM	MC68030	TEXT
ENDR	MC68040	TITLE
EQU	MC68851	TTL
EQUR	MC68881	USER
ERROR	MC68EC030	WARNING
EVEN	MC68EC040	XDEF
EXITM	MCCPU32	XREF

Appendice D • Instructions

Introduction

ASM reconnaît toutes les instructions du 68000 au 68040, ainsi que celles des 68881-2 et 68851 et du CPU32.

Il admet des synonymes pour les Bcc, DBcc, Scc, TRAPcc :

- **HS** est reconnu comme CC.
- **LO** est reconnu comme CS.
- **ZE** est reconnu comme EQ.
- **NZ** est reconnu comme NE.

BLO équivaut à BCS, SHS équivaut à SCC.

Il admet aussi **DBRA** comme synonyme de DBF.

ASM émule certaines instructions dans leur forme correcte pour leur exécution :

- ADD (resp. SUB, CMP) est transformé en ADDI (resp. SUBI, CMPI) si la source est immédiate (#i), ou en ADDA (resp. SUBA, CMPA) si la destination est un registre d'adresse.
- AND (resp. OR, EOR) est transformé en ANDI (resp. SUBI, EORI) si la source est immédiate (#i).
- MOVE est transformé en MOVEA si la destination est un registre d'adresse, et MOVEA en MOVE si ce n'est pas le cas.
- MOVEM demande une liste de registres composée de la forme Rn [/Rn ...] et de la forme Rm - Rn, R étant un registre de donnée ou d'adresse avec m>n. Rn peut aussi être un EQUR. MOVEM comprend aussi les notations:
 - reg où reg est un REG.
 - #i où i est le masque des registres de la liste.
 - #reg où reg est un REG.

Instructions MPU

Nom	00	10	20	30	40	32
A						
ABCD	✓	✓	✓	✓	✓	✓
ADD	✓	✓	✓	✓	✓	✓
ADDA	✓	✓	✓	✓	✓	✓
ADDI	✓	✓	✓	✓	✓	✓
ADDQ	✓	✓	✓	✓	✓	✓
ADDX	✓	✓	✓	✓	✓	✓
AND	✓	✓	✓	✓	✓	✓
ANDI	✓	✓	✓	✓	✓	✓
ANDI CCR	✓	✓	✓	✓	✓	✓
ANDI SR	✓	✓	✓	✓	✓	✓
ASL, ASR	✓	✓	✓	✓	✓	✓

	B					
Bcc (2)	✓	✓	✓	✓	✓	✓
BCHG	✓	✓	✓	✓	✓	✓
BCLR	✓	✓	✓	✓	✓	✓
BFCHG			✓	✓	✓	✓
BFCLR			✓	✓	✓	✓
BFEXTS			✓	✓	✓	✓
BFEXTU			✓	✓	✓	✓
BFFF0			✓	✓	✓	✓
BFINS			✓	✓	✓	✓
BFSET			✓	✓	✓	✓
BFTST			✓	✓	✓	✓
BGND						✓
BKPT		✓	✓	✓	✓	✓
BSET	✓	✓	✓	✓	✓	✓
BTST	✓	✓	✓	✓	✓	✓
	C					
CALLM			✓			
CAS			✓	✓	✓	
CAS2			✓	✓	✓	
CHK	✓	✓	✓	✓	✓	✓
CHK2			✓	✓	✓	
CLR	✓	✓	✓	✓	✓	✓
CMP	✓	✓	✓	✓	✓	✓
CMPA	✓	✓	✓	✓	✓	✓
CMPI	✓	✓	✓	✓	✓	✓
CMPM	✓	✓	✓	✓	✓	✓
CMP2			✓	✓	✓	
CPUSH (1)					✓	
	D					
DBcc	✓	✓	✓	✓	✓	✓
DIVS	✓	✓	✓	✓	✓	✓
DIVSL			✓	✓	✓	✓
DIVU	✓	✓	✓	✓	✓	✓
DIVUL			✓	✓	✓	✓
	E					
EOR	✓	✓	✓	✓	✓	✓
EORI	✓	✓	✓	✓	✓	✓
EORI CCR	✓	✓	✓	✓	✓	✓
EORI SR (1)	✓	✓	✓	✓	✓	✓
EXG	✓	✓	✓	✓	✓	✓
EXT	✓	✓	✓	✓	✓	✓
EXTB			✓	✓	✓	✓
	I					

ILLEGAL	✓	✓	✓	✓	✓	✓
	J					
JMP	✓	✓	✓	✓	✓	✓
JSR	✓	✓	✓	✓	✓	✓
	L					
LEA	✓	✓	✓	✓	✓	✓
LINK	✓	✓	✓	✓	✓	✓
LPSTOP						✓
LSL, LSR	✓	✓	✓	✓	✓	✓
	M					
MOVE	✓	✓	✓	✓	✓	✓
MOVEA	✓	✓	✓	✓	✓	✓
MOVE F CCR		✓	✓	✓	✓	✓
MOVE T CCR	✓	✓	✓	✓	✓	✓
MOVE F SR (2)	✓	✓	✓	✓	✓	✓
MOVE T SR (1)	✓	✓	✓	✓	✓	✓
MOVE USP (1)	✓	✓	✓	✓	✓	✓
MOVE16					✓	
MOVEC (1)		✓	✓	✓	✓	✓
MOVEM	✓	✓	✓	✓	✓	✓
MOVEP	✓	✓	✓	✓	✓	✓
MOVEQ	✓	✓	✓	✓	✓	✓
MOVES (1)		✓	✓	✓	✓	✓
MULS	✓	✓	✓	✓	✓	✓
MULU	✓	✓	✓	✓	✓	✓
	N					
NBCD	✓	✓	✓	✓	✓	✓
NEG	✓	✓	✓	✓	✓	✓
NEGX	✓	✓	✓	✓	✓	✓
NOP	✓	✓	✓	✓	✓	✓
NOT	✓	✓	✓	✓	✓	✓
	O					
OR	✓	✓	✓	✓	✓	✓
ORI	✓	✓	✓	✓	✓	✓
ORI CCR	✓	✓	✓	✓	✓	✓
ORI SR (1)	✓	✓	✓	✓	✓	✓
	P					
PACK			✓	✓	✓	
PEA	✓	✓	✓	✓	✓	✓
	R					
RESET (1)	✓	✓	✓	✓	✓	✓
ROL, ROR	✓	✓	✓	✓	✓	✓
ROXL, ROXR	✓	✓	✓	✓	✓	✓
RTD		✓	✓	✓	✓	✓

RTE (1)	✓	✓	✓	✓	✓	✓
RTM			✓			
RTR	✓	✓	✓	✓	✓	✓
RTS	✓	✓	✓	✓	✓	✓
	S					
SBCD	✓	✓	✓	✓	✓	✓
Scc	✓	✓	✓	✓	✓	✓
STOP (1)	✓	✓	✓	✓	✓	✓
SUB	✓	✓	✓	✓	✓	✓
SUBA	✓	✓	✓	✓	✓	✓
SUBI	✓	✓	✓	✓	✓	✓
SUBQ	✓	✓	✓	✓	✓	✓
SUBX	✓	✓	✓	✓	✓	✓
SWAP	✓	✓	✓	✓	✓	✓
	T					
TAS	✓	✓	✓	✓	✓	✓
TBLS, TBLSN						✓
TBLU, TBLUN						✓
TRAP	✓	✓	✓	✓	✓	✓
TRAPcc			✓	✓	✓	✓
TRAPV	✓	✓	✓	✓	✓	✓
TST	✓	✓	✓	✓	✓	✓
	U					
UNLK	✓	✓	✓	✓	✓	✓
UNPK		✓	✓	✓	✓	

(1) : ces instructions sont privilégiées.

(2) : pour les instructions Bcc, cc vaut :

RA, SR, CC, CS, EQ, GE, GT, HI, LE, LS, LT, MI, NE, PL, VC, VS.

(3) : pour les DBcc, Scc, TRAPcc, cc vaut :

CS, EQ, F, GE, GT, HI, LE, LS, LT, MI, NE, PL, T, VC, VS.

Instructions MMU

Elles sont toutes en mode superviseur.

Il n'y a aucune des ces instructions sur le 68EC030.

Nom	30	40	51
PBCC(1)			✓
PDBCC(1)			✓
PFLUSH(2) (3)	✓		
PFLUSHA(2) (3)	✓		
PFLUSH (2) (4)		✓	
PFLUSHN (4)		✓	
PFLUSHA (2) (3)		✓	
PFLUSHAN (3)		✓	
PFLUSH (2)			✓
PFLUSHA (2)			✓
PFLUSHS			✓
PFLUSHR			✓
PLOADR/W (3)	✓		✓
PMOVE (2)	✓		✓
PRESTORE			✓
PSAVE			✓
PSCC(1)			✓
PTESTR (2) (4)	✓	✓	✓
PTESTW (2) (4)	✓	✓	✓
PTRAPCC(1)			✓
PVALID			✓

(1) : pour Bcc, PDBcc, PScc, PTRAPcc, cc vaut :

BS, LS, SS, AS, WS, IS, GS, CS, BC, LC, SC, AC, WC, IC, GC, CC

(2) : ces instructions, quoique de nom identique, sont différentes pour chaque processeur.

(3) : ces instructions n'existent pas dans les versions 68EC030, 68EC040.

(4) : ces instructions ne doivent pas être utilisées sur le 68EC040.

Instructions FPU

Il n'y a aucune de ces instructions sur les 68EC040 et 68LC040.

Elles se trouvent toutes, en revanche, sur les 68881 et 68882.

S/D signale que les formes FxS et FxD existent (sur 68040).

Nom	81/2	40	S/D
FABS	✓	✓	✓
FACOS	✓		
FADD	✓	✓	✓
FASIN	✓		
FATAN	✓		
FATANH	✓		
FBcc (2)	✓	✓	
FCMP	✓	✓	
FCOS	✓		
FCOSH	✓		
FDBcc (2)	✓	✓	
FDIV	✓	✓	✓
FETOX	✓		
FETOXM1	✓		
FGETEXP	✓		
FGETMAN	✓		
FINT	✓		
FINTRZ	✓		
FLOG10	✓		
FLOG2	✓		
FLOGN	✓		
FLOGNP1	✓		
FMOD	✓		
FMOVE	✓	✓	✓
FMOVECR	✓		
FMOVEM	✓	✓	
FMUL	✓	✓	✓
FNEG	✓	✓	✓
FNOP	✓	✓	
FREM	✓		
FRESTORE (1)	✓	✓	
FSAVE (1)	✓	✓	
FSCALE	✓		
FScc (2)	✓	✓	
FSGLDIV	✓	✓	

FSGLMUL	✓	✓	
FSIN	✓		
FSINCOS	✓		
FSINH	✓		
FSQRT	✓	✓	✓
FSUB	✓	✓	✓
FTAN	✓		
FTANH	✓		
FTENTOX	✓		
FTRAPcc (2)	✓	✓	
FTST	✓	✓	
FTWOTOX	✓		

(1) : ces instructions sont privilégiées.

(2) : pour les FBcc, FDBcc, FScc, FTRAPcc, cc vaut :

EQ, NE, GT, NGT, GE, NGE, LT, NLT, LE, NLE, GL, NGL,
GLE, NGLE, OGE, OGT, OLE, OLT, OR, UEQ, UGE, UGT,
ULE, ULT, UN, F, T, SF, ST, SEQ, SNE

Appendice E • Messages d'erreur

Message	Explication
Argument value must be >=0	Cette directive demande un opérande positif.
Ascii too long	Le nombre en ASCII dépasse 4 caractères.
Bad register list	Ce n'est pas une liste de registres.
Bit number must be >=0 and <=63	La valeur de l'opérande de gauche d'un BCHG/BCLR/BSET/BTST, n'est pas comprise entre 0 et 63.
Bitfield offset must be >=0 and <=31	La valeur de l'offset d'un champ de bits n'est pas comprise entre 0 et 31.
Bitfield width must be >=1 and <=32	La taille d'un bitfield n'est pas comprise entre 1 et 32.
BKPT # must be >=0 and <=7	La valeur de l'opérande de BKPT n'est pas comprise entre 0 et 7.
Byte size forbidden	L'octet est interdit (généralement pour un mode d'adressage en An).
Can't allocate : xxx	Il ne reste plus de mémoire libre. xxx est Label, File, Expression, MACRO expansion, IFcc, REPT, Memory block, Module, Debug, Binary block.
Can't close file	Impossible de fermer le fichier.
Can't compute equ	Un equ ne peut faire partie d'un calcul.
Can't create file	Impossible de créer le fichier.
Can't get drive	Impossible de trouver le lecteur.
Can't get path	Impossible de trouver le chemin.
Can't open file	Impossible d'ouvrir le fichier.
Can't read file	Impossible de lire le fichier.
Can't record any more expression	Plus de mémoire.
Can't seek in file	Impossible de se déplacer dans le fichier.
Can't set drive	Impossible de changer de lecteur.
Can't set path	Impossible de changer de chemin.
Can't understand operand	Erreur générique quand aucune précision n'est possible.
Can't write file	Impossible d'écrire dans le fichier.
Couldn't load include file	Impossible de charger le fichier à inclure.
D16 is gt than word	La valeur du déplacement sur 16 bits est plus grande qu'un mot.

D8 is gt than byte	La valeur du déplacement sur 8 bits est plus grande qu'un octet.
Data forbidden in BSS or OFFSET section	On ne peut générer de données dans une section BSS ou OFFSET.
Displacement greater than byte	La distance de branchemennt est supérieure à un octet.
Displacement greater than word	La distance de branchemennt est supérieure à un mot.
Ea can only be shifted by 1	Les modes d'adressage différents de Dn ne peuvent être décalés que de 1.
ELSE outside IFCC	Un ELSE est rencontré en dehors d'un IFCC.
ENDIF outside IFCC	Un ENDIF est rencontré en dehors d'un IFCC.
ENDM outside MACRO	Un ENDM est rencontré en dehors d'une déclaration de MACRO.
ENDR outside REPT	Un ENDR est rencontré en dehors d'un REPT.
EQUATE not found in MACRO	Un nom d'EQU entre <> est rencontré mais n'est pas encore défini.
EQU must equate to explicit register	EQU ne prend que des noms de registres (D0~D7, A0~A7, SP, PC, ZA0~ZA7, ZPC) et des noms d'EQU.
EQU not allowed	Le calcul fait intervenir un EQU.
EXITM outside MACRO	Un EXITM est rencontré en dehors d'une déclaration de MACRO.
Expecting : " or '	Il manque un guillemet ou une apostrophe.
Expecting : '#'	Il manque un signe dièse.
Expecting : '('	Il manque une parenthèse ouvrante.
Expecting : ')'	Il manque une parenthèse fermante.
Expecting : '+'	Il manque un plus.
Expecting : ','	Il manque une virgule.
Expecting : '-'	Il manque un signe moins.
Expecting : ':'	Il manque un deux-points.
Expecting : '>' after equate in MACRO	Il manque un > après un nom d'EQU dans une MACRO.
Expecting : '{' or '}'	Il manque une accolade.
Expecting : An	Il manque un registre d'adresse.
Expecting : An or PC	Il manque un registre d'adresse ou une indication de PC relatif.
Expecting : Cache line	Il faut une ligne de cache.

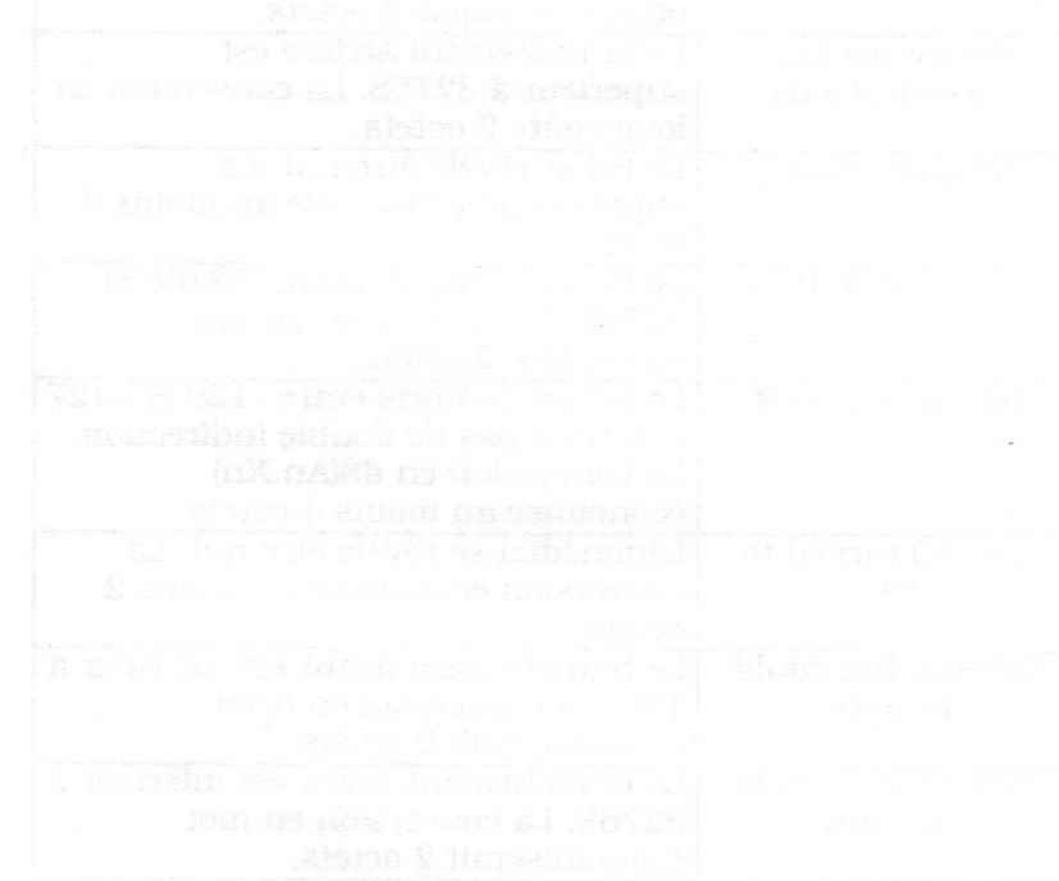
Expecting : Control register	Il faut un nom de registre de contrôle.
Expecting : d16(An)	Il faut le mode d'adressage d16(An)
Expecting : Dn	Il manque un registre de donnée.
Expecting : Dn or -(An)	Il manque un registre de donnée ou une prédecrémentation.
Expecting : End of operand	Il reste du texte après que l'opérande entier a été assemblé.
Expecting : Ending " or '	Il manque un guillemet ou une apostrophe fermant.
Expecting : Extended ('.x') size	Seule la taille extended est autorisée avec cet opérande.
Expecting : Filename	Cette directive exige un nom de fichier.
Expecting : FPcr (FPCR, FPSR, FPIAR)	Il faut un registre de contrôle FPU.
Expecting : FPIAR with An	Le mode d'adressage An doit correspondre au FPIAR.
Expecting : Fpu register	Il manque un registre flottant.
Expecting : Label	Cette directive exige un label.
Expecting : Operand	Il manque un opérande.
Expecting : Packed ('.p') size	Seule la taille packed est autorisée avec un k-factor.
Expecting : PMMU function code	Ce n'est pas un code de fonction PMMU.
Expecting : Register list	Il faut une liste de registres.
Expecting : Register or separator ('/' or '-')	Une liste ne peut contenir que des registres et les séparateurs / et -.
Expecting : Scale	Il manque un scale après un registre d'index.
Expecting : Scale or size	Il manque une taille ou un scale après un registre d'index.
Expecting : Size (.w or .L)	Il manque une taille après un registre d'index.
Expecting : Starting " or '	Il manque un guillemet ou une apostrophe ouvrant.
Expecting : Word size instruction with this ea	Cette instruction n'admet que le mot comme taille avec ce mode d'adressage.
Expecting : Xn	Il manque un registre d'index.
Expecting binary operator	Il manque un opérateur binaire.

Expression must evaluate	Les opérandes de l'expression ne sont pas tous définis.
Forbidden binary operator	L'opérateur binaire se trouve à un endroit interdit.
Forbidden ea	Ce mode d'adressage est interdit.
Forbidden ea : xxx	Le mode d'adressage xxx est interdit pour cette instruction. xxx peut être : Dn, An, (An), (An)+, -(An), d16(An), d8(An), d16(PC), d8(PC), Memory indirection, Abs.w, Abs.l, #i, Scale (MPU<CPU32).
Forbidden ea: 68020	Ce mode d'adressage n'est autorisé qu'à partir du 68020.
Forbidden mnemonic: xxx	Le mnémonique du processeur xxx est interdit pour le processeur courant. xxx est 68010, 68020, 68030, 68040, CPU32, MMU, FPU.
Forbidden reference for object output	Le XREF est utilisé dans un endroit incompatible avec le type d'objet généré.
Forbidden relative expression	Le résultat doit être constant.
Forbidden size	La taille spécifiée pour ce mnémonique est interdite.
Forbidden size with Dn ea	La taille de l'instruction est incompatible avec le mode d'adressage Dn.
Forbidden size with fpn ea	La taille de l'instruction est incompatible avec l'utilisation de registre FPU.
Forward reference to SET variable	L'expression fait référence à un SET défini postérieurement.
FPU coprocessor required	Les constantes et les calculs flottants demandent un FPU.
Garbage following string in IFC or IFNC	Des caractères se trouvent après une chaîne dans un opérande de IFC ou IFNC.
Illegal character : xxx	Un caractère illégal a été rencontré : (<32) : compris entre 0 et 31. (>127) : compris entre 128 et 255. opcode : différent de A~Z,a~z,0~9, _, ?. label : différent de A~Z,a~z,0~9, _, ?, ..
Illegal character for radix	Le nombre contient des caractères qui ne correspondent pas à sa base numérique.

Illegal fpurom operand	La valeur de l'opérande de fpurom0 est interdite.
Illegal operator	Opérateur illégal.
Illegal type combination	L'opérateur est incompatible avec ses opérandes.
Illegal variable type	Type de variable illégal.
In register list : Multi-defined register	Un registre est défini en double dans une liste de registres.
Incompletely enclosed MACRO parameter	Des caractères se trouvent après un '>'.
Invalid MACRO parameter	Un caractère différent de 1~9, a~z, A~Z, @, # et \ se trouve derrière un \
K-factor must be >=-64 and <=63	La valeur du k-factor n'est pas comprise entre -64 et 63.
Label already used with : xxx	xxx est SET, EQU, MACRO.
Local name forbidden	Cette directive n'accepte pas de labels locaux.
MACRO name matches opcode name	Un nom de macro est identique à celui d'un opcode.
MMUSR forbidden with PMOVEFD	PMOVEFD n'admet pas le MMUSR.
Negative shift value	Un décalage négatif est impossible.
Nested MACRO declarations	On ne peut déclarer une MACRO dans une MACRO.
No ENDIF matching with IFCC	La fin du source est atteinte sans qu'un ENDIF ait été rencontré.
No ENDM matching with MACRO	La fin du source est atteinte sans qu'un ENDM ait été rencontré.
No ENDR matching with REPT	La fin du source est atteinte sans qu'un ENDR ait été rencontré.
Not yet implemented	N'existe pas encore dans cette version d'ASM.
Null Bcc.b is forbidden	Un branchement sur 8 bits ne peut pointer immédiatement en dessous de lui-même.
Number too big	Le nombre contient trop de chiffres.
Opcode not found	L'opcode n'est ni un mnémonique, ni une directive, ni un nom de macro.
Opcode size forbidden	La taille de l'opcode spécifiée est interdite.
Operand value is gt than byte	La valeur de l'opérande est plus grande qu'un octet.

Operand value is gt than word	La valeur de l'opérande est plus grande qu'un mot.
Org forbidden in BSS or OFFSET section	On ne peut générer de BSS ou d'OFFSET en absolu.
Output type must be set before any code generation / XDEF / XREF	Le type de fichier généré est changé alors que du code a été généré ou des XDEFS / XREFS ont déjà été rencontrés.
Overflow	Le calcul dépasse un long.
Padding byte inserted for even alignment	Un octet nul a été inséré pour pouvoir subséquemment être aligné.
PMMU function code must be $>=0$ and $<=7$	La valeur du code de fonction n'est pas comprise entre 0 et 7.
Privileged instruction	L'instruction est privilégiée.
Quick value must be $>=1$ and $<=8$	La valeur de l'opérande de gauche d'un ADDQ/SUBQ n'est pas comprise entre 1 et 8.
Redefining : xxx	xxx est EQU, SET, EQUR, REG, FEQU, FSET, FEQUR, FREG, MACRO.
Relative required	Le calcul ne fait pas intervenir d'adresse.
REPT counter must be >0	Le compteur de REPT ne peut être négatif ou nul.
Shift value must be $>=1$ and $<=8$	La valeur de l'opérande de gauche d'un LSL/LSR, ASL/ASR, ROL/ROR, ROXL/ROXR, n'est pas comprise entre 1 et 8.
SHIFTM outside MACRO	Un SHIFTM est rencontré en dehors d'une déclaration de MACRO.
Syntax error	Erreur de syntaxe.
This version of ASM will not run on current MPU	Cette version d'ASM a été prévue pour un processeur plus puissant.
Too many commas in parenthesis	Il y a plus de 2 virgules dans les parenthèses.
Too many globals (>65535) for object output	Il y a plus de 65535 globales dans le fichier objet.
Too many nested MACRO calls	L'empilement d'appels de MACROS a saturé la mémoire disponible.
Too many operands	Il y a trop d'opérandes.
Too many Xn's	Il y a plus d'1 registre d'index.
TRAP # must be $>=0$ and $<=15$	La valeur de l'opérande de TRAP n'est pas comprise entre 0 et 15.

Unbalanced : xxx	Le deuxième guillemet, apostrophe, parenthèse, crochet est absent.
Unexpected : close parenthesis	La parenthèse fermante est mal placée.
Unexpected : end of expression	L'expression n'est pas complète.
Unexpected : end of source	La fin du source est rencontrée dans une déclaration de macro ou en assemblage conditionnel.
Unexpected : open parenthesis	La parenthèse ouvrante est mal placée.
Unknown function	Cette fonction est inconnue
Unknown label	Le label est inconnu.
User error	La directive FAIL a été rencontrée.
Wrong operand number	Le nombre d'opérandes de la fonction est faux
XDEFed symbol not found	Un symbole XDEFé n'existe pas.
Zero divide	Division par zéro.



Appendice F • Messages d'avertissement

Optimisations

Message	Explication
0(An) turned to (An)	Le déplacement sur 16 bits se révèle être nul. La conversion en (An) économise 2 octets.
Abs.I turned to Abs.w	L'adresse absolue est comprise entre -32768 et 32767. La conversion en mot économise 2 octets.
ADD/SUB #i turned to ADDQ/SUBQ	i est compris entre 1 et 8. La conversion économise au moins 2 octets.
ADDA/SUBA #i turned to LEA	i est compris entre -32768 et 32767. L'exécution est plus rapide.
Backward Bcc turned to byte	Le branchement arrière est supérieur à 128. La conversion en octet économise 2 octets.
Backward Bcc turned to long	Le branchement arrière est supérieur à 32768. La conversion en long coûte 2 octets.
Bd suppressed	Le Bd se révèle être nul. La suppression économise au moins 2 octets.
Bd turned to Bd.w	Le Bd est compris entre -32768 et 32767. La conversion en mot économise 2 octets.
Bd turned to d8	Le Bd est compris entre -128 et +127 et il n'y a pas de double indirection. La conversion en d8(An,Xn) économise au moins 4 octets.
CMPI #0 turned to TST	L'immédiat se révèle être nul. La conversion économise au moins 2 octets.
Forward Bcc could be byte	Le branchement avant est inférieur à 128. La conversion en octet économiserait 2 octets.
Forward Bcc could be word	Le branchement avant est inférieur à 32768. La conversion en mot économiserait 2 octets.

LEA turned to ADDQ/SUBQ	Dans LEA d16(An), An avec les 2 An identiques, d16 est compris entre 1 et 8. La conversion économise 2 octets.
Long d16 turned to Bd.l	Le déplacement sur 16 bits est inférieur à -32768 ou supérieur à 32767. La conversion en long coûte 2 octets.
LSL #1 turned to ADD	LSL #1,Dn est converti en ADD Dn,Dn. L'exécution est plus rapide.
MOVE.L #i,Dn turned to MOVEQ	i est compris entre -128 et 127. La conversion en MOVEQ économise 4 octets.
MOVE.W #0,Dn turned to CLR Dn	L'immédiat se révèle être nul. La conversion en CLR économise 2 octets.
MOVEA #0 turned to SUBA	L'immédiat se révèle être nul. La conversion en SUBA économise au moins 2 octets.
MOVEA.L #i turned to MOVEA.W	i est compris entre -32768 et 32767. La conversion en mot économise 2 octets.
Null Bcc.B turned to NOP	Un branchement sur 8 bits ne peut pointer immédiatement en dessous. Il est remplacé par un NOP (taille id).
Od suppressed	Le Bd se révèle être nul. La suppression économise au moins 2 octets.
Od turned to Od.w	L'Od est compris entre -32768 et 32767. La conversion en mot économise 2 octets.
ROXL #1 turned to ADDX	ROXL #1,Dn est converti en ADDX Dn,Dn. L'exécution est plus rapide.

Divers

Message	Explication
Bit number will be modulo 8	Une instruction de manipulation de bit a comme premier opérande un n° de bit supérieur à 8 et comme second opérande une adresse.
Bit number will be modulo 32	Une instruction de manipulation de bit a comme premier opérande un n° de bit supérieur à 32 et comme second opérande un registre.
Empty register list	Une liste de registres dans un MOVEM ou un FMOVEM est vide. C'est normalement inutile.
Implicit comment	Du texte se trouve après une instruction, sans symbole de commentaire (; ou *).
MOVEQ value will be sign extended (>=- 128 && <0)	Le premier opérande d'un MOVEQ est compris entre 128 et 255
Software emulated FPU mnemonic	Ce mnémonique FPU n'existe pas sur ce processeur mais peut être géré par une procédure d'exception.
Software emulated FPU size	Cette taille FPU n'existe pas sur ce processeur mais peut être gérée par une procédure d'exception.
Spurious XDEF	XDEF rencontré alors que le programme généré n'est pas objet.
Spurious XREF	XREF rencontré alors que le programme généré n'est pas objet.
Stack pointer will be odd	Un LINK ou un RTD ont comme opérande une valeur impaire. C'est normalement une erreur.
Unrecommended mnemonic for EC MPU	Ce mnémonique MMU n'est pas censé être utilisable sur la version EC du processeur.
Unused XREF	Un XREF déclaré est inutilisé.

Appendice G • Spécificités Atari

Formats de sortie

ASM permet de générer deux types de fichiers objets et un type d'exécutable. Ce dernier est au format GEMDOS ATARI et peut contenir des symboles de 8 caractères (option D) ou de 22 caractères (option X). Le format d'objet DRI (option l2) ne permet que des symboles de 8 caractères, des XREFs relatifs PC sur 16 bits et des XREFs immédiats sur 16 ou 32 bits. Le format d'objet Pure C permet des symboles non limités en taille, des XREFs relatifs PC sur 16 bits ou 32 bits et des XREFs immédiats sur 16 ou 32 bits.

Gestion des sections

Le système de l'Atari reconnaît trois types de section : TEXT, DATA et BSS. Vous pouvez déclarer ces sections à tout endroit du source, sachant qu'ASM concatènera toutes les sections de même type.

 TEXT
nop
DATA
dc.w 1
BSS
ds.w 1
TEXT
moveq #0,d0
DATA
dc.w 2
BSS
ds.w 2
END

est équivalent à :

TEXT
nop
moveq #0,d0
DATA
dc.w 1,2
BSS
ds.w 3
END

Compatibilité avec d'autres assembleurs

Devpac (Hisoft) :

Les directives suivantes n'existent pas dans ASM :

COMMENT, RADIX, FORMAT

IIF doit être transformé en IF ... ENDIF.

R0-R15 doivent être définis en EQUR.

Les options d'assemblage d'ASM sont compatibles avec les options de GenST2/Genst3/Gen, sauf pour les nouvelles syntaxes d'options (comme DEBUG, etc...) qui ne sont pas reconnues. Leur ancienne syntaxe est reconnue, en revanche.

L'option c#+/- (où # est le nombre de caractères significatifs et +/- active / désactive la différenciation majuscule/minuscule) ne comprend que + ou -.

ASM ne génère pas de code objet au format GST ou Lattice C. Il génère en revanche le format Pure C.

Devpac accepte n'importe quel ordre dans un mode d'adressage avec Base ou Outer Displacement. Assemble, lui, s'en tient à l'ordre prévu par Motorola, qui évite bien des confusions.

Metacomco Assembler (Metacomco)

Aucune adaptation à faire.

Pure Assembler (Pure Software)

Tous les points devant les directives doivent être supprimés.

= est synonyme de EQU et non de SET.

*=<valeur> est à remplacer par ORG <valeur>.

ALINE et FLINE n'existent pas (il suffit d'écrire DC.W).

Il n'y a pas dans ASM l'équivalent de MODULE/ENDMOD, et les sections ne peuvent être nommées, ni numérotées.

COMM <label>,<taille> est à remplacer par :

<label> DS.B <taille>

Les ENDM utilisés pour signaler la fin d'un bloc de REPT doivent être transformés en ENDR.

GLOBL n'existe pas (il faut utiliser XREF ou XDEF suivant les cas).

IFB et IFNB sont à remplacer par IFD et IFND.

IFF est à remplacer par IFEQ.

IF1 et IF2 doivent être enlevés (si, étonnamment, ils sont présents).

LCOMM <label>,<taille> doit être remplacé par :

.<label> DS.B <taille>

Les noms des paramètres formels des macros doivent être remplacés par '\n°', où n° est le numéro du paramètre (à partir de 1). Les labels locaux déclarés par LOCAL doivent être convertis en <@\label>.

MacMAC (Atari Corporation)

Tous les points devant les directives doivent être supprimés, et ces dernières tabulées normalement dans le champ opération.

== et :: doivent être remplacés par XDEF.

ASSERT doit être transformé en IF ... FAIL ... ENDIF.

COMM n'existe pas (à émuler avec DS).

GLOBL/EXTERN doivent être remplacés par des XREF ou XDEF correspondant.

GOTO n'existe pas.

IIF doit être transformé en IF ... ENDIF.

INIT doit être remplacé par une série de DC.

NLIST doit être remplacé par NOLIST.

UNDEFMAC n'existe pas.

Il n'y a pas d'EQUates locaux.

Il n'y a pas de caractère d'échappement ('\') dans les chaînes de caractères.

R0-R15 doivent être définis en EQUR.

Les crochets ('[' et ']') dans les expressions arithmétiques doivent être remplacés par des parenthèses.

L'évaluateur d'ASM respectent les priorités d'opérateurs!

Les fonctions spéciales de l'évaluateur (^) n'existent pas.

Les noms des paramètres formels des macros doivent être remplacés par '\n°', où n° est le numéro du paramètre (à partir de 1).

\~ dans une macro doit être remplacé par \@.

\! dans une macro doit être remplacé par \0 (qui n'est d'ailleurs pas le dixième paramètre!).

ASM ne comprend pas le 6502 :=).

Gfa Assembleur (Micro-Application)

Le texte doit être exporté en ASCII.

Tous les points devant les directives doivent être supprimés.

SETCLU n'existe pas (heureusement!).

ABS est à remplacer par OFFSET.

SECTION possède une syntaxe légèrement différente.

COMM, LOCAL, LMODE n'existent pas.

Les noms des paramètres formels des macros doivent être remplacés par '\n°', où n° est le numéro du paramètre (à partir de 1). Les labels locaux déclarés avec \~ doivent être convertis en <@\label>.

IIFcc doit être transformé en IFcc ... ENDIF.

ASSERT <condition> doit être remplacé par :

IFNE <condition>

FAIL

ENDIF

INIT est remplacé par CARGS.

PRINTER est à transformer en LISTCHAR.

MLIST/NOMLIST sont à convertir en OPT -M+/-.

SPACE est à remplacer par SPC.

Profilal ST (Micro-Application)

EQU ne peut servir à définir une chaîne de caractères.

Le symbole définissant une variable locale est '.' et non '\'.

<@label> est à remplacer par <label> set <valeur>.

SLABEL, ENDS, ILABEL, INPUT, START n'existent pas.

REPEAT ... UNTIL est à convertir en REPT <nombre> ...

ENDR.

Turbo-Assembler (Sigma Software)

Le texte doit être exporté en ASCII.

GNU Assembler (GNU Software)

Hélas, presque tout le code est à réécrire. Désolé.

Appendice H • Bibliographie

Française

Le microprocesseur 68000 et sa programmation, Patrick Jaulent, Editions Eyrolles 1983.

Mise en oeuvre du 68000, Catherine Vieillefond, Editions Sybex.

Microprocesseurs 68020 68030 et leurs coprocesseurs, Patrick Jaulent, Luc Baticle, Pascal Pillot, Editions Eyrolles 1989.

Mise en oeuvre du 68030, Catherine Vieillefond, Editions Sybex 1989.

Anglaise

Programmer's Reference Manual (M68000PM/AD REV. 1), Motorola 1992.

MC68030 Enhanced 32-bit Microprocessor User's Manual 2nd Edition (MC68030 UM/AD REV. 1), Motorola 1989.

MC68881/MC68882 Floating-Point Coprocessor Users's Manual (MC68881UM/AD REV. 1), Motorola 1987.

Paged Memory Management Unit User's Manual 2nd Edition (MC68851UM/AD REV. 1), Motorola 1989.

68000 assembly language programming 2nd edition, Leventhal, Hawkins, Kane, Cramer, Osborne McGraw-Hill 1986.

Appendice I • Table ASCII

◊	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	P	`	p	ç	É	á	ã	í	ó	ú	ñ	é
1	↑	!	!	!	A	Q	a	q	ü	æ	í	õ	ý	ü	þ	‡
2	◊	2	"	2	B	R	b	r	é	€	ó	ø	x	ø	g	≥
3	◊	3	#	3	C	S	c	s	â	ô	ú	ø	í	ç	π	≤
4	◊	4	\$	4	D	T	d	t	ä	ö	ñ	œ	æ	í	Σ	†
5	◊	5	X	5	E	U	e	u	à	ò	ñ	Œ	œ	ł	σ	J
6	▀	6	&	6	F	V	f	v	â	û	à	À	پ	ш	÷	
7	▀	7	'	7	G	W	g	w	ç	ù	o	Ã	і	پ	τ	z
8	√	8	(8	H	X	h	x	ê	ÿ	ë	õ	ı	ı	ő	°
9	⊕	9)	9	I	Y	i	y	ë	ö	”	”	پ	ı	θ	•
A	‡	a	*	:	J	Z	j	z	è	Ü	”	”	ø	ø	ø	.
B	„	ß	+	;	K	[k	{	ï	ç	ķ	č	č	đ	đ	đ
C	ؒ	c	,	<	L	\	l		î	£	կ	ç	ç	ҹ	ҹ	ҹ
D	ؒ	d	-	=	M]	m	}	ì	ყ	ı	ø	ø	ø	ø	ø
E	ؒ	x	.	>	N	^	n	~	ă	þ	«	ø	ø	ø	ø	ø
F	ؒ	ß	/	?	O	_	o	Δ	ă	ƒ	»	”	ј	ø	ø	-

Appendice J • Index

~ 16
^ 16
! 16
!= 16
\$ 14
% 14
& 16; 49
(An) 19
(An)+ 19
(BdSz,An,XnSz*Sc) 20
(BdSz,PC,XnSz*Sc) 20
* 10; 15
** 48
+ 15
, 12
- 14; 15
-(An) 19
. 11
.sizeof 39
/ 15
68008 44
68010 44
68020 44
68030 45
68040 45
68300 44
68330 46
68340 46
68851 46
68881 46
68882 46
68EC000 44
68HC000 44
< 16
<< 15
<= 16
<> 16; 49
= 16; 31
== 16
> 16; 49
>= 16
>> 15; 49
@ 14
@<>
ABS 16
abs.l 20
abs.w 20
ACC 9
ACOS 16
ALIGN 28
An 19
apostrophe 14
APP 9
ASCII 36
ASCIIL 36
ASCIIZ 36
ASIN 16
ASM 7
assemblage conditionnel 53
ATAN 16
ATANH 16
avertissemens 25
B 11
B+/- 8
base displacement 20
BIN 7
binaire 14
BSS 22; 29
C 40
C+/- 8
CARGS 40
CNOP 28
CODE 29
codes de contrôle 42
commentaire 12
complément à 1 16
complément à 2 15
coprocesseur 25
corps de la macro 47
COS 16
COSH 16
CPU32 46
Ctrl-C 8
Ctrl-Q 8
Ctrl-S 8
D 11
D+/- 8
d16(An) 19

d16(PC) 19	EXITM 48
d8(An,XnSz) 19	EXP 16
d8(An,XnSz*Sc) 20	expansion de macro 47
d8(PC,XnSz) 20	EXPORT 43
d8(PC,XnSz*Sc) 20	FAIL 30
D= 8	FEQU 33
DATA 22; 29	FEQUR 34
DBRA 62	fin de macro 47
DC 35	FOPT 26
DCB 35	FPUROM 16
débogage source 25	FREG 34
décimal 14	FSET 33
déclaration de macro 47	GTP 9
directive 11	guillemet 14
Dn 19	hexadécimal 14
DRI 23; 24; 25	HS 62
DS 37	I<nom> 8
E 8; 14	IBYTES 27
E+/- 8	ID= 26
ELSE 55	IF 54
ELSEIF 56	IFC 55
ELSEIFC 56	IFD 55
ELSEIFD 56	IFEQ 54
ELSEIFEQ 56	IFGE 54
ELSEIFGE 56	IFGT 54
ELSEIFGT 56	IFLE 54
ELSEIFLE 56	IFLT 54
ELSEIFLT 56	IFNC 55
ELSEIFNC 56	IFND 55
ELSEIFND 56	IMPORT 43
ELSEIFNE 56	INCBIN 27
en-tête de la macro 47	INCDIR 27
END 30	include 24; 26
ENDC 56	INT 16
ENDIF 56	INTRZ 16
ENDM 48	K= 26
ENDR 57	L 11
EPROM 9	L0/1/2 8
EQU 31	Label 7
equate 31	LIST 41
EQUR 32	LISTCHAR 42
ERROR 30	listing 25; 41
Espace blanc 7	LLEN 42
étiquette 7	LO 62
ETOX 16	LOG10 16
ETOXM1 16	LOG2 16
EVEN 28	LOGN 16