

RASM: Roudoudou's Z80 Assembler V0.117

Édouard BERGÉ (Roudoudou)

24 décembre 2019

Table des matières

1	Introduction	4
1.1	Fonctionnalités	4
2	Comportement Général	5
2.1	Invocation	5
2.2	Controle des noms de fichier	5
2.3	Exportation de symboles	5
2.4	Inclusions et importations	6
2.5	Dépendances	7
2.6	Options de compatibilité	7
2.7	Options de développement	7
2.8	Autres options	7
3	Syntaxe Générale	8
3.1	Commentaires	8
3.2	Labels	8
3.3	Mnémoniques Z80	8
3.3.1	Registres IX,IY	8
3.3.2	Syntaxes des instructions non documentées	9
3.3.3	Syntaxes spéciales	9
3.4	Directives mémoires	9
3.4.1	Directive ORG	9
3.4.2	ALIGN	10
3.4.3	LIMIT	10
3.4.4	PROTECT	10
3.5	Données	10
3.5.1	DB, DEFB, DM, DEFM	10
3.5.2	DEFW	11
3.5.3	DEFI	11
3.5.4	DEFR	11
3.5.5	DEFS	11
3.5.6	STR	12
3.5.7	CHARSET	12
3.5.8	Opérateur \$	12

4	Expressions	14
4.1	Alias, et Variables	14
4.1.1	Variables statiques ou alias	14
4.1.2	Valeurs dynamiques	14
4.2	Valeurs littérales	14
4.2.1	Valeurs Numériques	14
4.2.2	Caractères autorisés	15
4.3	Opérateurs de calcul	15
4.4	Priorité d'exécution des opérateurs	15
5	Préprocesseur	17
5.1	Debug et assertion	17
5.1.1	PRINT	17
5.1.2	FAIL	17
5.1.3	STOP	17
5.2	Directives conditionnelles	17
5.2.1	ASSERT	17
5.2.2	IF, IFNOT	18
5.2.3	IFDEF, IFNDEF	18
5.2.4	UNDEF	18
5.2.5	IFUSED, IFNUSED	18
5.2.6	SWITCH	18
5.3	Boucles et macros	19
5.3.1	REPEAT	19
5.3.2	WHILE, WEND	19
5.3.3	Définition d'une MACRO	20
5.3.4	Labels Locaux	21
5.3.5	Labels de proximité	21
5.4	Structures de données	22
5.4.1	STRUCT	22
5.4.2	SIZEOF	22
5.5	Durée d'un bloc	23
6	Importation et compression	24
6.1	Importation de fichiers	24
6.1.1	INCLUDE	24
6.1.2	INCBIN	24
6.1.3	Importation de fichiers multiples	24
6.1.4	Variantes	25
6.1.5	Fichiers Audio	25
6.2	Compression	25
6.2.1	Portions compressées	25
6.2.2	Inclusion de fichiers compressés	26
7	Directives et options spécifique à l'Amstrad CPC	27
7.1	Manipulation des Banques	27
7.1.1	Préfixe BANK	27
7.1.2	Préfixe PAGE	27
7.1.3	Préfixe PAGESET	27
7.2	Export de fichiers DSK et entêtes AMSDOS	28
7.2.1	Entête AMSDOS	28
7.2.2	Directive SAVE	28

7.3	Cartouches et Snapshots	29
7.3.1	BUILDSNA	29
7.3.2	BANK	29
7.3.3	RUN	30
7.4	Directives spécifiques aux snapshots	30
7.4.1	BANKSET	30
7.4.2	SETCPC	30
7.4.3	SETCRTC	31
7.4.4	BREAKPOINT	31
7.4.5	Options d'export	31
7.5	Gestion des couleurs du CPC+	31
7.5.1	GET_R, GET_G, GET_B	31
7.5.2	SET_R, SET_G, SET_B	32
7.6	Directives obsolètes	32
7.6.1	NOCODE	32
7.6.2	WRITE DIRECT	32
7.6.3	LIST, NOLIST, LET	32
8	Directives et options spécifique au ZX	33
8.1	Selection des Banques	33
8.1.1	Directive BUILDZX	33
8.2	Options spécifiques	33
9	Compilation	34
9.1	Compiler RASM	34
9.1.1	Linux	34
9.1.2	Windows (Visual Studio)	34
9.1.3	Dos/Windows 32 (Watcom)	34
9.1.4	MacOS	34
9.1.5	MorphOS	34
9.2	Intégrer RASM	34
9.2.1	Récupération des erreurs et symboles	35
A	Coloration syntaxique	36
A.1	Coloration syntaxique dans VIM	36
B	Opcodes Z80	37
B.1	Instructions principales	37
B.2	Instructions étendues (ED)	37
B.3	Instruction de manipulation de bits (CB)	38
B.4	Instructions IX (DD)	38
B.5	Instruction de manipulations de bits IX (DDCB)	39
B.6	Instructions IY (FD)	40
B.7	Instruction de manipulation de bits IY (FDCB)	41
C	Durée des opcodes du Z80 sur CPC	42
D	Licenses	44

1 Introduction

Lors de la création de ma démo CRTC³, les assembleurs existants ont tous rapidement montré leurs limites, que ce soit en termes de rapidité d'assemblage ou de fonctionnalités. En effet, le projet final pesait plus de 250'000 mots (hors commentaires), 35'000 labels et 60'000 expressions. Il fallait donc en priorité un assembleur qui soit rapide et lui ajouter au minimum des fonctionnalités que n'ont pas les autres : support natif des cartouches, disquettes ou snapshot Amstrad, espaces mémoires multiples, calculs flottants, compressions les plus courantes intégrées, afin de s'éviter par exemple de fastidieux export/imports de symboles, etc.

C'est ainsi qu'est né RASM. Ce n'était pas mon premier assembleur, j'en avais déjà conçu un 18 ans auparavant, mais il était trop limité. J'en ai néanmoins repris certains principes, comme l'assemblage en une seule passe. Rasm utilise des concepts éprouvés (arbres de Merkel, nombreux caches, allocations groupées) et surtout une conception linéaire, la récursivité tuant les performances.

Résultat : RASM présente des performances d'assemblage très élevées en conditions réelles, il est très rapide, même sur de gros projets.

Aujourd'hui Rasm est utilisé dans de nombreux projets d'envergure :

- Ghost'n Goblins par Golem13
- Arkos Tracker II par Targhan de façon intégrée au logiciel

1.1 Fonctionnalités

- compilation ultra-rapide
- export au format binaire, cartouche, disquettes, snapshot...
- export/import des symboles
- intégration des compresseurs les plus courants
- contrôle automatique des zones mémoires contre l'écrasement
- possibilité d'utiliser des variables en plus des constantes traditionnelles
- espaces mémoires illimités où les labels/variables sont partagées
- absolument toutes les instructions Z80 sont supportées
- macros, code conditionnel, boucles illimités, labels locaux, switch/case
- calculs internes en double précision et arrondi correct
- compatibilité optionnelle avec Maxam ou AS80
- gestion dédiée des ROM et extensions mémoires
- importation de fichiers audio, avec conversion en DMA list (CPC+)
- calcul de la durée des instructions

Rasm est diffusé sous licence MIT (voir Annexe D pour l'ensemble des licences).

La présente documentation est maintenue par Stéphane Sikora, merci de faire remonter vos remarques à l'adresse sikooglerasm@gmail.com.

2 Comportement Général

RASM est conçu pour être simple d'utilisation, tout en offrant beaucoup de souplesse. Par défaut, il nomme les fichiers produits de façon cohérente, recherche les fichiers par chemin relatif, détermine la quantité de mémoire à enregistrer ou même créer un fichier cartouche si les banques ROM ont été sélectionnées lors de l'assemblage, autorise l'utilisation de plusieurs directives ORG, etc.

Dans la mesure du possible, RASM essaie d'afficher des messages d'erreurs afin de vous orienter vers la solution syntaxique convenable.

RASM va d'abord pré-traiter le fichier à assembler pour enlever les espaces superflus, les commentaires, vérifier les quotes, que les caractères utilisés soient conformes et enfin transformer certaines instructions en d'autres pour convenance interne.

Par exemple les opérateurs Maxam XOR, AND, OR et MOD seront convertis en un seul caractère approchant la syntaxe du C.

Enfin, il est à noter que RASM n'intègre pas d'éditeur, ce n'est pas un IDE, mais uniquement l'assembleur. Selon les plateformes, vous pouvez utiliser votre éditeur préféré, si possible dans lequel il est possible de configurer la colorisation syntaxique pour l'assembleur z80.

2.1 Invocation

La syntaxe de base est :

```
RASM.exe <fichier à assembler> [options]
```

Par exemple,

```
RASM.exe myfile.asm
```

produira le fichier `rasmoutput.bin`

2.2 Contrôle des noms de fichier

Les options suivantes permettent de contrôler le nom des fichiers générés :

- -o <radix de fichier> : définit un nom commun pour chaque type de fichier en sortie, quel que soit son type (.bin, .sym, etc.). La valeur par défaut est "rasmoutput"
- -ob <nom de fichier .bin> : définit le nom complet pour le fichier binaire en sortie
- -os <nom de fichier .sym> : définit le nom complet pour le fichier des symboles
- -oa : donner aux fichiers de sortie (cpr,bin,sna) le même nom que le source
- -no : désactive la génération de fichiers en sortie.

2.3 Exportation de symboles

Ces options permettent de générer un fichier avec les valeurs et adresses associées aux symboles du code assemblé :

- -s : exporter les symboles au format RASM
- -sp : exporter les symboles au format PasmO
- -sw : exporter les symboles au format Winape
- -sl : option additionnelle qui permet d'exporter aussi les symboles locaux aux macros ou boucles de répétition.
- -sv : option additionnelle qui permet d'exporter aussi les variables.
- -sq : option additionnelle qui permet d'exporter aussi les alias EQU.

- -sx : exporte les symboles pour certains émulateurs ZX, en précisant la banque où le symbole est logé ((<bank>:<adresse>))
- -sa : option équivalente à -sl -sv -sq

Par exemple :

```
RASM.exe test -o grouik -s
Pre-processing [test.asm]
Assembling
Write binary file grouik.bin (25 bytes)
Write symbol file grouik.sym (10 bytes)
```

Le fichier .sym est de la forme :

```
LABEL1 #0 B0
LABEL2 #1 B0
LABEL3 #2 B0
LABEL4 #4 B0
```

Avec l'option -sp, le fichier de symboles est le suivant :

```
LABEL1 EQU 00000H
LABEL2 EQU 00001H
LABEL3 EQU 00002H
LABEL4 EQU 00004H
```

Avec l'option -sw, le fichier de symboles est compatible avec Winape :

```
LABEL1 #0
LABEL2 #1
LABEL3 #2
LABEL4 #4
```

Il est possible de définir des blocs de code (directives NOEXPORT) pour lesquels l'exportation de symbole est désactivée.

2.4 Inclusions et importations

Il est possible d'inclure des fichiers sources ou binaires à l'aide de directives comme INCBIN ou READ. Par défaut les fichiers sont recherchés dans le repertoire local, de facon relative, mais il est possible de définir un ou plusieurs répertoires pour aller chercher les fichiers à inclure. Il est possible aussi d'importer ou définir des symboles :

- -I <include directory> : Définit un répertoire dans lequel rechercher des fichiers à inclure. Il est possible d'utiliser plusieurs fois cette option. Les fichiers sont recherchés dans l'ordre des répertoires indiqués.
- -D <var>=<valeur > : option pour définir une variable depuis la ligne de commande. Exemple : -DTRUC=1 va définir la variable TRUC à 1.
- -l <fichier label> : importer un fichier de labels pour l'assemblage au format RASM, Sjasn, Pasmu ou Winape. La détection du format est automatique. Il est possible de cumuler plusieurs fichiers de symboles en cumulant les options -l :

```
RASM.exe test -l import1.sym -l import2.sym -l import3.sym
```

2.5 Dépendances

- -depend=make : Exporte toutes les dépendances du programme sur une seule ligne (pour une utilisation en makefile).
- -depend=list : Exporte toutes les dépendances du programme, une par ligne.

Si un nom de fichier binaire est spécifié (option -ob) alors il sera ajouté en première position dans la liste des dépendances.

2.6 Options de compatibilité

- -m : compatibilité Maxam
 - Les calculs réalisés en entiers 16 bits non signés avec un mauvais arrondi
 - Les comparaisons se font avec l'opérateur = simple
 - La priorité des opérateurs est simplifiée (voir tableau)
- -ass : compatibilité AS80
 - calculs réalisés en entiers 32 bits avec un mauvais arrondi
 - Les déclarations de type DEFB,DEFW,DEFI ou DEFR multiples ont pour référence d'adresse l'adresse du premier octet produit et non l'adresse de l'octet courant. Cette spécificité d'AS80 fait que deux DEFB ne produisent pas la même chose qu'un seul DEFB avec deux valeurs quand la référence \$ est utilisée.
 - Les paramètres des macros ne sont plus protégés par les chevrons
 - La directive MACRO s'utilise après le nom de la macro et non avant
- -uz : compatibilité UZ80
 - Les calculs sont réalisés en entiers 32 bits avec un mauvais arrondi
 - Les paramètres des macros ne sont plus protégés par les chevrons
 - La directive MACRO s'utilise après le nom de la macro et non avant

2.7 Options de développement

Ces options sont moins courantes ,mais peuvent parfois etre utiles pour le debug.

- -v : mode verbeux, affiche des informations et statistiques sur l'assemblage
- -void : permet de forcer l'utilisation de (void) pour les macros sans parametre (??)
- -d : produit des informations lors du pré-processing
- -a : produit des informations lors de l'assemblage
- -n : affiche les licences tierces

2.8 Autres options

Il existe d'autres options, qui concernent des architectures spécifiques, en particulier le CPC. La section 7.4.5 en particulier. Enfin, l'option -help donne un récapitulatif de toutes les arguments qui peuvent être passés a RASM.

3 Syntaxe Générale

La syntaxe supportée par RASM se veut souple, laissant de nombreuses libertés au développeur.

Par exemple, il n'est pas nécessaire d'indenter vos sources avec Rasm, si ce n'est pour des raisons esthétiques. Il n'est pas nécessaire non plus d'utiliser le caractère ' : ' pour définir un label, bien qu'il soit possible de le faire. Les deux points sont simplement ignorés.

De même, les fichiers peuvent être au format windows ou linux, une conversion interne et transparente est réalisée. Enfin, Rasm n'est pas sensible à la casse, toutes les lettres sont converties en majuscules en interne. Aussi, ne soyez pas surpris de ne voir que des majuscules dans les messages d'erreur.

Nous verrons que cette liberté implique certaines ambiguïtés qui seront précisées dans la suite. La seule contrainte forte imposée par RASM est qu'il est interdit de créer un label qui ait le même nom qu'une directive ou instruction Z80.

Dans cette partie nous verrons la syntaxe générale d'un code en z80 comme on en le retrouve avec les assembleurs classiques. Dans les parties suivantes, nous verrons les aspects spécifiques de RASM.

3.1 Commentaires

La saisie de commentaire sous Rasm est classique et précédée du caractère point-virgule. Tous les caractères suivants sont ignorés jusqu'au prochain retour chariot. Il est aussi possible d'utiliser la syntaxe du C, à savoir le double slash (//) pour débiter un commentaire, ainsi que les symboles /* et */ pour délimiter des commentaires sur plusieurs lignes .

3.2 Labels

Les labels permettent de nommer une adresse mémoire liée à du code ou des données. Dans certains assembleurs, ils sont suivis de deux points, dans d'autres, non. RASM supporte les deux syntaxes.

Il existe des labels spéciaux. Par exemple tout label qui commence par le préfixe BRK ou @BRK génère en outre un point d'arrêt (cf section 7.4.4). Nous verrons aussi par la suite qu'il est possible de définir des labels locaux à des boucles ou macros.

```
| ld HL,monlabel  
| call aFunction  
| aFunction:  
| ; ...  
| ret  
| monlabel db 0
```

3.3 Mnémoniques Z80

Toutes les instructions du Z80 - documentées et non documentées - sont supportées.

3.3.1 Registres IX,IY

L'adressage 8 bits des registres d'index IX et IY se fait indifféremment avec LX, IXL ou XL, etc.

```
| ld A,IXL
```

Par ailleurs, les instructions complexes avec IX et IY s'écrivent de la façon suivante :

```
| res 0,(IX+d),A  
| bit 0,(IX+d),A  
| sll 0,(IX+d),A  
| rl 0,(IX+d),A  
| rr 0,(IX+d),A
```


3.3.2 Syntaxes des instructions non documentées

```
out (<byte>),a
in a,(<byte>)
in 0,(c)
in f,(c)
sll <register>
sl1 <registre>
```

3.3.3 Syntaxes spéciales

Il existe des raccourcis qui permettent de rendre le code plus compact et plus lisible.

- PUSH/POP multi-arguments
 - PUSH BC,DE,HL → PUSH BC : PUSH DE : PUSH HL
 - POP HL,DE,BC → POP HL : POP DE : POP BC
- NOP répétitif
 - nop 4 → nop : nop : nop : nop
- LD Complexe
 - LD BC,BC → LD B,B : LD C,C
 - LD BC,DE → LD B,D : LD C,E
 - LD BC,HL → LD B,H : LD C,L
 - LD DE,BC → LD D,B : LD E,C
 - LD DE,DE → LD D,D : LD E,E
 - LD DE,HL → LD D,H : LD E,L
 - LD HL,BC → LD H,B : LD L,C
 - LD HL,DE → LD H,D : LD L,E
 - LD HL,HL → LD H,H : LD L,L
- LD Complexe avec IX,IY
 - LD HL,(IX+n) → LD H,(IX+n+1) : LD L,(IX+n)
 - LD HL,(IY+n) → LD H,(IY+n+1) : LD L,(IY+n)
 - LD DE,(IX+n) → LD D,(IX+n+1) : LD E,(IX+n)
 - LD DE,(IY+n) → LD D,(IY+n+1) : LD E,(IY+n)
 - LD BC,(IX+n) → LD B,(IX+n+1) : LD C,(IX+n)
 - LD BC,(IY+n) → LD B,(IY+n+1) : LD C,(IY+n)
 - LD (IX+n),HL → LD (IX+n+1),H : LD (IX+n),L
 - LD (IY+n),HL → LD (IY+n+1),H : LD (IY+n),L
 - LD (IX+n),DE → LD (IX+n+1),D : LD (IX+n),E
 - LD (IY+n),DE → LD (IY+n+1),D : LD (IY+n),E
 - LD (IX+n),BC → LD (IX+n+1),B : LD (IX+n),C
 - LD (IY+n),BC → LD (IY+n+1),B : LD (IY+n),C
- Syntaxes alternatives
 - EXA → EX AF,AF'

3.4 Directives mémoires

3.4.1 Directive ORG

ORG <logical address>[,<physical address>]

La directive ORG permet d'indiquer l'adresse de départ du code et des données à assembler. Rasm permet l'utilisation de plusieurs ORG au sein d'un même espace mémoire, mais il contrôle qu'aucune zone de code écrite ne se chevauche : il n'autorise pas de ré-écrire sur les mêmes adresses mémoire.

```
ORG #8000
RET
; bytecode output:
; #8000: #C9
```

Si vous avez besoin de générer plusieurs morceaux de code à la même adresse, vous avez deux possibilités. Soit vous utilisez le deuxième paramètre de la directive `ORG` pour écrire ce code ailleurs, soit vous pouvez créer à tout moment un nouvel espace mémoire avec la directive `BANK` 7.3.2

```
ORG #8000,#1000
label: JP label
; bytecode output:
; #1000: #C3,#00,#80
```

3.4.2 ALIGN

`ALIGN <boundary>[,fill]`

Si l'adresse d'écriture du code en cours n'est pas un multiple de la valeur d'alignement, on augmente l'adresse en conséquence. Par défaut, cette instruction ne produit pas d'octet sur la sortie (les espaces de travail sont initialisés avec la valeur 0). Mais il est possible de préciser la valeur de remplissage avec les second paramètre. Par exemple :

```
ORG #8001
ALIGN 2 ; align code on even address (#8002)
ALIGN 256,#55 ; align code on high byte (#8100)
; #8002-#80FF is filled with #55
```

3.4.3 LIMIT

`LIMIT <address boundary>`

Cette directive sert à imposer une limite plus basse à l'écriture de l'assemblage. Par défaut, la limite est de 65535 mais on peut avoir besoin de ne pas dépasser une certaine valeur. Pour protéger une zone définie, il vaut mieux utiliser la fonction `PROTECT`.

3.4.4 PROTECT

`PROTECT <start address>,<end address>`

Empêcher l'écriture de données dans la zone début/fin de l'espace mémoire courant.

3.5 Données

3.5.1 DB, DEFB, DM, DEFM

```
DEFB <value1>[,<value2>,...]
DEFM <value1>[,<value2>,...]
```

Cette directive prend un ou plusieurs paramètres et génère une suite d'octets représentative des paramètres en entrée. La valeur peut être une valeur littérale, une formule dont le résultat sera éventuellement arrondi et tronqué ou même une chaîne de caractères dont chaque caractère produira un octet en sortie égale à sa valeur ASCII, sauf si la directive `CHARSET` a été utilisée pour redéfinir ces valeurs. On pourra indifféremment utiliser `DB`, `DM`, `DEFB` ou `DEFM`. Par exemple le code suivant produira la chaîne 'Roudoudou' : (le 'u' correspond au code ASCII #75)

```
org #7500
label:
defb 'r'-'a'+'A','oudoud',#6F,hi(label)
```

See CHARSET directive for altering the way strings are interpreted.

3.5.2 DEFW

```
DEFW, DW <value1>[,<value2>,...]
```

La directive DEFW (ou son raccourci DW) prend un ou plusieurs paramètres et génère une suite de mots (deux octets, en little endian) représentative des paramètres en entrée. Les valeurs peuvent être une valeur littérale, une formule dont le résultat sera éventuellement arrondi et tronqué. Cette directive ne supporte pas une chaîne de plusieurs caractères comme valeur.

Exemple :

```
| DEFW mylabel1,mylabel2,'a'+#100
```

3.5.3 DEFI

```
DEFI <valeur1>[,<valeur2>,...]
```

Cette directive prend un ou plusieurs paramètres et génère une suite de double-mots (4 octets) représentative des paramètres en entrée. Les valeurs peuvent être une valeur littérale, une formule dont le résultat sera éventuellement arrondi et tronqué. Cette directive ne supporte pas une chaîne de plusieurs caractères comme valeur.

3.5.4 DEFR

```
DEFR <reel1>[,<reel2>,...]
```

La directive DEFR (ou DR) prend un ou plusieurs paramètres et génère une suite de nombre réels (5 octets chaque) compatibles avec le firmware des Amstrad CPC.

Exemple :

```
| defr 5/12, 0.5, sin(90)
```

3.5.5 DEFS

```
DEFS, DS <repetition>[,<value>[,<repetition>,...]]
```

Cette directive génère une suite d'octets répétitive. Si la valeur de remplissage est omise alors zéro sera utilisé par défaut. Si la valeur de répétition est nulle alors aucun octet ne sera produit. Il est possible de déclarer plusieurs suites de répétition avec le même DEFS mais les premières répétitions sera toujours interprétées comme ayant une valeur à répéter.

Exemples :

```
defs 5,8,4,1 ; #08,#08,#08,#08,#08,#01,#01,#01
defs 5,8,4   ; #08,#08,#08,#08,#08,#00,#00,#00,#00
defs 5       ; #00,#00,#00,#00,#00
```

3.5.6 STR

```
STR 'string1'[, 'string2'...]
```

Directive similaire à DEFB, sauf que le dernier caractère de chaque chaîne aura son bit 7 forcé à 1 (opération OR #80 sur le dernier octet). Les deux lignes suivantes produiront les mêmes octets :

```
| str 'roudoudou'  
| defb 'roudoudo', 'u'+128
```

3.5.7 CHARSET

```
CHARSET  
CHARSET 'string', <value>  
CHARSET <code>, <value>  
CHARSET <start>, <end>, <value>
```

La directive permet de redéfinir des valeurs aux caractères assemblés entre quotes selon quatre formats :

- 'chaîne', <valeur> : Le premier caractère de la chaîne aura pour nouvelle valeur la <valeur>. Le caractère suivant <valeur>+1 et ainsi de suite pour tous les caractères de la chaîne.
- <code>, <valeur> : Attribuer au caractère de valeur ASCII <code> la valeur <valeur>.
- <début>, <fin>, <valeur> : Attribuer aux caractères de valeur ASCII <début> à <fin> une valeur incrémentale en partant de <valeur>.
- aucun paramètre : réassigne à tous les caractères leur valeur ASCII par défaut.

Cette fonction est compatible Winape.

Par exemple, une simple redéfinition :

```
| CHARSET 'T', 't' ; 'T' chars will be translated as 't'  
| DEFB 'tT' ; #74 #74
```

Ou encore de redéfinir par intervalle :

```
| CHARSET 'A', 'Z', 'a' ; Change all uppercases to their respective lowercases  
| DEFB 'abcdeABCDE' ; #61, #62, #63, #64, #65, #61, #62, #63, #64, #65
```

Il est possible de redéfinir des caractères non consécutifs :

```
| CHARSET 'turndiskheo ', 0  
| DEFB 'there is no turndisk'  
| ; #00, #08, #09, #02, #0B, #05, #06, #0B, #03, #0A, #0B, #00, #01, #02, #03, #04, #05, #06, #07
```

3.5.8 Opérateur \$

```
| org #8000  
| defw $, $
```

est équivalent à :

```
| defw #8000, #8002
```

En mode de compatibilité AS80, le même code serait équivalent à

```
| defw #8000,#8000
```

Cependant, lorsque \$ est utilisé avec la directive ORG, il fait référence à l'adresse physique et non l'adresse logique :

```
| ORG #8000,#1000
| defw $ ; #8000 is written in #1000
| ORG $ ; ORG considers the physical address (#1002)
| defw $ ; #1002 is written in #1002
| ; bytecode output:
| ; #1000: #00,#80,#02,#10
```

4 Expressions

4.1 Alias, et Variables

Il est possible de créer un nombre illimité d'alias avec la directive EQU. Ces alias ne peuvent pas être modifiés une fois qu'ils sont définis. Les variables au contraire peuvent être modifiées.

4.1.1 Variables statiques ou alias

`<alias> EQU <replacement string>`

Création d'un alias. Quand l'assembleur rencontrera l'alias dans une expression, le texte sera remplacé par celui défini lors du EQU. Un test de récursivité infini est réalisé à la création de l'alias.

Exemple :

```
tab1 EQU #8000
tab2 EQU tab1+#100
ld HL,tab2
```

4.1.2 Valeurs dynamiques

Les constantes déclarées avec EQU ne peuvent pas être modifiées une fois définies. Il est possible avec RASM de définir des variables, avec la syntaxe suivante :

```
myvar=5
LET myvar=5 ; Winape compatible Variables are used for numeric values only.
```

Rasm autorise un nombre illimité de variables. Voici quelques exemples :

```
dep=0
repeat 16
ld (IX+dep),A
dep=dep+8
rend

ang=0
repeat 256
defb 127*sin(ang)
ang=ang+360/256
rend
```

4.2 Valeurs littérales

4.2.1 Valeurs Numériques

Rasm interprète les valeurs numériques de la façon suivante :

- En décimal si la valeur commence par un chiffre.
- En binaire si la valeur commence par un % ou 0b.
- En octal si la valeur commence par un @.
- En hexadécimal si la valeur commence par un #, un \$, 0x ou se termine par un h.
- En valeur ascii si un caractère unique est entre quotes.
- En valeur associée à une variable ou un label, si la littérale commence par une lettre ou un '@' pour les labels locaux.

- Le symbole \$ utilisé seul indique l'adresse de début de l'instruction en cours. Lors d'un define type DEFB, DEFW, DEFI, DEFR ou DEFS, l'adresse courante est celle de l'élément en cours. Un DEF* produira les mêmes données que vous utilisiez plusieurs arguments à la suite ou bien plusieurs DEF*.

Attention, le caractère & est réservé pour l'opérateur AND.

4.2.2 Caractères autorisés

Entre quotes, tous les caractères sont autorisés, à vos risques et périls concernant la conversion ASCII vers l'Amstrad. En dehors des quotes, vous pourrez utiliser toutes les lettres, tous les chiffres, le point, l'arobas, les parenthèses, le dollar, les opérateurs plus, moins, multiplié, divisé, le pipe, circonflexe, le pourcent, le dièse, le paragraphe, les chevrons et les deux types de quotes ainsi que les caractères échappés \t \n \r \f \v \b \0 . Ceci ne s'applique pas avec la directive PRINT.

4.3 Opérateurs de calcul

Rasm utilise un moteur d'expression simplifié à priorités multiples (comme le C). Il supporte les opérateurs et fonctions suivants :

*	multiplication	/	division
+	addition	−	soustraction
^ ou XOR	opérateur logique OU Exclusif	% ou MOD	Reste de la division (Modulo)
& ou AND	opérateur logique ET	ou OR	opérateur logique OU
&&	opérateur booléen ET		opérateur booléen OU
<<	décalage à gauche (multiplication par 2 ⁿ)	>>	décalage à droite (division par 2 ⁿ)
hi()	poids fort de l'entier 16 bits	lo()	poids faible de l'entier 16 bits
sin()	calcul de sinus	cos()	calcul de cosinus
asin()	calcul d'arc-sinus	acos()	calcul d'arc-cosinus
atan()	calcul d'arc-tangente		
int()	conversion en nombre entier	frac()	recupere la partie fractionnaire
floor()	conversion à l'entier entier inférieur	abs()	valeur absolue
ln()	logarithme népérien	log10()	logarithme base 10
exp()	exponentielle	sqrt()	racine carrée
==	égalité (= en mode Maxam)	!= ou <>	différent de
<=	inférieur ou égal	>=	supérieur ou égal
<	inférieur	>	supérieur

Rasm fait tous ses calculs internes en nombre flottant double précision. Un arrondi correct est réalisé en fin de chaine de calcul pour les besoins en nombres entiers. Si l'évaluation d'une expression aboutit à une erreur de calcul, le résultat de l'évaluation sera forcé à 0.

4.4 Priorité d'exécution des opérateurs

La tableau suivant indique la prévalence des opérateurs : plus elles est basse, plus la priorité est élevée.

Opérateur	Prévalence Rasm	Prévalence Maxam
()	0	0
!	1	464
* / %	2	464
+ -	3	464
<< >>	4	464
< <= == => > !=	5	664
& AND	6	464
OR	7	464
^ XOR	8	464
&&	9	6128
	10	6128

5 Préprocesseur

5.1 Debug et assertion

5.1.1 PRINT

`PRINT 'string',<variable>,<expression>`

Cette directive permet d'écrire du texte, variables ou expressions dans la console, au cours de l'assemblage. Il est impératif que les variables que l'on souhaite afficher existent au préalable. Concernant les valeurs numériques, l'affichage par défaut est un nombre flottant mais il est possible de formater les variables en les préfixant par des tags :

- `{hex}` afficher la variable en hexadécimal. Si la variable vaut moins de `#FF` alors l'affichage sera forcé sur deux chiffres. Si la variable vaut moins de `#FFFF` alors l'affichage sera forcé sur quatre chiffres. Au dessus il n'y aura pas d'extra-zéros.
- `{hex2}`, `{hex4}`, `{hex8}` pour forcer l'affichage quel que soit la valeur, sur 2, 4 ou 8 chiffres.
- `{bin}` afficher la variable en binaire. Si la variable vaut moins de `#FF` alors l'affichage sera forcé sur 8 bits. Si la variable vaut moins de `#FFFF` alors l'affichage sera forcé sur 16 bits. Un pré-traitement enlève les 16 bits supérieurs de la valeur 32 bits au cas où tous les bits sont à 1 (nombre négatif).
- `{bin8}`, `{bin16}`, `{bin32}` pour forcer l'affichage quel que soit la valeur, sur 8, 16 ou 32 bits.
- `{int}` afficher en décimal, nombre entier.

5.1.2 FAIL

`FAIL 'string',<variable>,<expression>`

Cette directive est similaire à `PRINT`, mais elle produit une erreur et arrête l'assemblage.

5.1.3 STOP

Arrêter l'assemblage. Aucun fichier ne sera écrit.

5.2 Directives conditionnelles

RASM supporte un ensemble de directives qui permettent d'influer sur l'assemblage en vérifiant des expressions conditionnelles. D'une manière générale, lorsque des variables sont utilisées dans ces expressions, il faut impérativement qu'elles soient définies au préalable.

5.2.1 ASSERT

`ASSERT <condition>[,text,text,text...]`

Vérifier une condition et arrêter l'assemblage si la condition est fausse. Le texte supplémentaire est envoyé vers la console dans ce cas.

```
| assert mygenend-mygenstart<#100  
| assert mygenend-mygenstart<#100, 'code is too big'
```

5.2.2 IF, IFNOT

```
IF <condition> ... [ELSE ...] ENDIF
IF <condition> ... [ELSEIF <condition> ...] ENDIF
IFNOT <condition> ... [ELSE, ... ] ENDIF
IFNOT <condition> ... [ELSEIF <condition> ...] ENDIF
```

Directive de test et définition de blocs pour le code conditionnel.

```
CODE_PRODUCTION=1
[...]
if CODE_PRODUCTION
    or #80
else
    print 'test version'
endif
```

5.2.3 IFDEF, IFNDEF

```
IFDEF <variable or label> ... [ELSE ... ] ENDIF
IFNDEF <variable or label> ... [ELSE ... ] ENDIF
```

Directive de test et définition de blocs pour le code conditionnel.

5.2.4 UNDEF

```
UNDEF <variable>
```

Permet de retirer la définition d'une variable. La condition pour une directive `IFDEF` sera alors évaluée à faux. Si la variable n'existe pas, la directive est sans effet.

5.2.5 IFUSED, IFUNUSED

```
IFUSED <variable or label> ... ENDIF
IFUNUSED <variable or label> ... ENDIF
```

Ces deux directives permettent de tester l'utilisation ou la non utilisation d'une variable ou d'un label, AVANT l'utilisation de la directive.

5.2.6 SWITCH

La syntaxe est similaire au switch/case en C, avec la particularité de pouvoir écrire plusieurs blocs `CASE` avec la même valeur, ce qui donne plus de souplesse au code conditionnel. Un bloc switch se termine avec la directive `ENDSWITCH`. Dans cet exemple, la chaîne 'BCE' sera produite.

```
myvar EQU 5

switch myvar
    nop ; outside any case, will never be evaluated
case 3
    defb 'A'
case 5
    defb 'B'
case 7
    defb 'C'
break
```

```
case 8
    defb 'D'
case 5
    defb 'E'
    break
default
    defb 'F'
endswitch
```

5.3 Boucles et macros

5.3.1 REPEAT

```
REPEAT <number of repetitions>[,counter] ... REND
REPEAT ... UNTIL <condition>
```

Répète un bloc d'instructions. On peut soit fixer un nombre de répétitions, soit utiliser le mode conditionnel avec la directive de fin de bloc UNTIL. Pour compatibilité avec Vasm, il est possible de finir un tel bloc par ENDREP ou ENDREPEAT

```
cnt=90
repeat
    defb 64*sin(cnt)
    cnt=cnt-4
until cnt<0
```

Dans le cas de répétition non conditionnelle, il est possible de spécifier une variable qui contiendra le numéro d'itération. Cette variable (ici 'cnt') n'a pas besoin d'exister, elle sera créée le cas échéant, au début de la boucle REPEAT.

```
repeat 10,cnt
    ldi
    print cnt
rend
```

La variable interne REPEAT_COUNTER contient le numéro de l'itération courante (en commençant par 1).

```
repeat 10
    ldi
    print repeat_counter
rend
```

5.3.2 WHILE, WEND

```
WHILE <condtion> ... wEND
```

Répète un bloc tant que la condition est évaluée a vrai. A tout moment, il est possible d'utiliser la variable interne WHILE_COUNTER pour récupérer le compteur de boucle.

```
cpt=10
while cpt>0
    ldi
    cpt=cpt-1
    print 'cpt=',cpt,' while_counter=',while_counter
wend
```

La boucle va s'exécuter 10 fois, avec le résultat suivant sur la sortie standard :

```
Pre-processing [while.asm]
Assembling
cpt= 9.00 while_counter= 1.00
cpt= 8.00 while_counter= 2.00
cpt= 7.00 while_counter= 3.00
cpt= 6.00 while_counter= 4.00
cpt= 5.00 while_counter= 5.00
cpt= 4.00 while_counter= 6.00
cpt= 3.00 while_counter= 7.00
cpt= 2.00 while_counter= 8.00
cpt= 1.00 while_counter= 9.00
cpt= 0.00 while_counter= 10.00
Write binary file rasmoutput.bin (20 bytes)
```

5.3.3 Définition d'une MACRO

```
MACRO <macro_name> [param1[,param2[,...]]]
...
MEND
```

Une macro est une façon d'étendre le langage, en définissant un bloc d'instructions, délimité par **MACRO** et **MEND** (ou **ENDM**), et qui pourra être inséré ultérieurement dans le code, par simple utilisation du nom de la macro. Les macros peuvent prendre des paramètres, il est ainsi possible de faire de l'assemblage conditionnel avec les macros : à chaque appel de macro, le code d'origine est inséré, avec substitution des paramètres. Ce n'est qu'ensuite que le code interprété de façon classique.

Voici un exemple pour une écriture longue distance générique (sauf pour B ou C) :

```
macro LDIXREG register,dep
    if {dep}<=-128 || {dep}>127
        push BC
        ld BC,{dep}
        add IX,BC
        ld (IX+0),{register}
        pop BC
    else
        ld (IX+{dep}},{register}
    endif
endef
mend
```

```
LDIXREG H,200
LDIXREG L,32
```

Utilisation d'une macro Attention ! Rasm ne peut pas savoir si vous voulez déclarer un label quand vous vous trompez dans l'écriture du nom d'une macro sans paramètre. Pour palier à ce défaut, vous pouvez ajouter un paramètre fictif "(void)" qui déclenchera une erreur si le nom de macro n'est pas connu.

De plus, si vous souhaitez forcer l'utilisation de cette syntaxe, il est possible de lancer RASM avec l'option **-void**. L'utilisation d'une macro sans paramètre provoquera une erreur.

```
MACRO withoutparam
nop
```

```
MEND
withoutparam (void) ; secured call of macro
```

Appel de macro avec parametres dynamiques Les parametres passés lors de l'appel d'une macro ne sont pas nécessairement des constantes, ils peuvent etre des expressions :

```
MACRO test myarg
    DEFB {myarg}
MEND
;Identique à 1,2:
REPEAT 2
    test repeat_counter
REND
;Identique à defb 1,1:
REPEAT 2
    test {eval}repeat_counter
REND
```

5.3.4 Labels Locaux

À l'intérieur d'une boucle (REPEAT/WHILE/UNTIL) ou d'une macro, il est possible d'utiliser des labels locaux de la même façon qu'avec l'assembleur intégré de Winape en préfixant le label par le caractère '@'. L'intérêt est que l'on peut réutiliser le même label à différents endroits du code, sans avoir à chercher des noms différents.

À chaque itération de boucle et ce pour chaque imbrication de boucle, un suffixe est ajouté au label local contenant la valeur hexadécimale du compteur interne de répétition. Il est ainsi possible d'appeler un label local à une répétition en dehors de la boucle, mais cet usage n'est pas conseillé.

Il est possible d'utiliser la valeur d'un label dans une commande (ORG par exemple) si et seulement si le label précède la directive.

En mode DAMS, il est possible d'utiliser la déclaration désuète d'un label en le préfixant d'un point. L'appel à ce label se fera sans le point du début. L'usage des labels de proximité est alors désactivé.

```
repeat 16
add hl,bc
jr nc,@no_overflow
dec hl
@no_overflow
rend
```

5.3.5 Labels de proximité

Hors d'une macro ou d'une boucle, les labels locaux héritent du label global qui précède. Exemple d'utilisation d'un label de proximité :

```
routine1
    add hl,bc
    jr nc,.no_overflow
    dec hl
.no_overflow

routine2
    add hl,bc
    jr nc,.no_overflow
    dec hl
```

```
.no_overflow
routine3
  xor a
  ld hl,routine1.no_overflow ; retrieve proximity label of routine1
  ld de,routine2.no_overflow ; of routine2
  sbc hl,de
```

5.4 Structures de données

5.4.1 STRUCT

```
STRUCT <prototype name> [,<variable name>]
...
ENDSTRUCT
```

```
; structure st1 created with two fields ch1 and ch2.
struct st1
  ch1 defw 0
  ch2 defb 0
endstruct
; Nested structures:
; metast1 is created with 2 sub-structures st1 called pr1 et pr2
struct metast1
  struct st1 pr1
  struct st1 pr2
endstruct
```

Quand {STRUCT} est utilisé avec 2 paramètres, RASM va créer une structure en mémoire, basée sur le prototype. Dans l'exemple ci dessous, une structure de type 'metast1' sera instanciée, avec pour nom 'mymeta' :

```
| struct metast1 mymeta
```

Exemple de récupération de l'adresse absolue d'un membre en utilisant le nom d'une structure déclarée :

```
| LD HL,mymeta.pr2.ch1
| LD A,(HL)
```

Exemple d'accès d'un membre avec un offset, en utilisant le nom du prototype :

```
| LD A,(IX+metast1.pr2.ch1)
```

5.4.2 SIZEOF

Le préfixe {SIZEOF} devant un nom de structure, de sous-structure ou même devant le champ d'une structure, permet de récupérer sa taille.

```
| LD A,{SIZEOF}metast1 ; LD A, 6
| LD A,{SIZEOF}metast1.pr2 ; LD A, 3
| LD A,{SIZEOF}metast1.pr2.ch1 ; LD A, 2
```

5.5 Durée d'un bloc

```
TICKER START,<var>
...
TICKER STOP,<var>
```

Cette directive calcule la durée d'un bloc d'instructions (délimité par TICKER START et TICKER STOP) et stocke le résultat dans une variable. Elle n'est pour le moment valable que pour CPC, la durée calculée est exprimée en "nombre de NOPs équivalents" (cf Annexe C), ce qui correspond à peu près à la durée en micro secondes. Cette directive est très pratique pour écrire et maintenir du code à durée fixe. Par exemple, si on veut faire un effet 'raster', il faut il faut s'assurer que l'on change de couleur à chaque ligne écran, soit toutes les 64 microsecondes :

```
ld hl,col_tab
ld bc,col_port ;#7fxx
out (c),c
ld d,20
loop:
TICKER START, cntline
ld a,(hl)
out (c),a
inc hl
TICKER STOP, cntline
ds 64-4-cntline
dec d
jr nz, loop
```

Comme on sait que les deux dernières instructions (DEC et JR) durent "4 NOPs", on rallonge la boucle la boucle de 60-cntline NOPs.

6 Importation et compression

6.1 Importation de fichiers

6.1.1 INCLUDE

```
INCLUDE 'file to read'
READ 'file to read'
```

Lire un fichier texte et l'intégrer au code source à l'emplacement de l'instruction de lecture. Le chemin relatif de lecture a pour racine l'emplacement du fichier dans lequel est l'instruction de lecture. Un chemin absolu s'affranchit de ce répertoire racine. Il n'y a pas de limite de récursivité en lecture, donc attention à ce que vous faites.

6.1.2 INCBIN

```
INCBIN 'file to read'[,offset[,size[,extended offset[,OFF]]]]
INCBIN 'file to read',REVERT
INCBIN 'file to read',REMAP,numcol
INCBIN 'file to read',VTILES,numtiles
INCBIN 'file to read',ITILES,width
```

Lire un fichier binaire. Les données lues seront directement injectées dans le binaire. Les paramètres optionnels sont compatibles avec la fonction INCBIN de Winape. L'offset n'est pas limité à 64Ko comme Winape. L'offset étendu est là pour compatibilité. Son usage est à éviter pour rester lisible.

- Il est possible de donner un offset négatif, relatif à la fin du fichier.
- Il est possible de donner une taille de fichier négative. La taille lue sera égale à la taille totale du fichier ajoutée à cette valeur négative. Pour tout lire sauf les 10 derniers octets, on précisera une taille de -10 dans la commande.
- Une taille nulle chargera tout le fichier.
- Paramètre OFF : Si on souhaite charger un fichier pour initialiser de la "mémoire" et qu'on souhaite assembler du code par dessus, on peut désactiver pour la lecture de ce fichier le contrôle d'écrasement.

Exemple :

```
ORG #4000
INCBIN 'makeraw.bin',0,0,0,OFF ; read in #4000, overwrite check is disabled

ORG #4001
DEFB #BB ; overwrite 2nd byte (in #4001) without error
```

Un autre exemple, dans lequel un fichier de 32K est lu en deux fois, par exemple dans deux banques de 16 différentes (cf 7.3.2)

```
bank n
incbin 'my32Kbfile.bin',0,16384
bank n+1
incbin 'my32Kbfile.bin',16384,16384
```

6.1.3 Importation de fichiers multiples

Il est possible d'utiliser INCBIN dans un bloc REPEAT, par exemple si l'on veut importer plusieurs fichiers, ici les fichier myfile1, myfile2, ... myfile10 :


```
REPEAT 10,cpt
INCBIN 'myfile{cpt}'
REND
```

6.1.4 Variantes

La forme avec **REVERT** permet d'importer un fichier binaire, mais dans le sens inverse.

Les variantes utilisant **REMAP**, **VTILES** et **ITILES** sont utiles pour importer des sprites.

ITILES permet d'importer des tiles de hauteur multiples de 8, en entrelaçant les lignes dans l'ordre suivant : 0,1,3,2,6,7,5,4. De plus les octets sont encodés en zig zag : pour les lignes paires, la séquence d'octets correspond aux pixels de gauche à droite, et de droite à gauche sur les lignes impaires. Le paramètre passé correspond à la largeur (en octet) des tiles.

6.1.5 Fichiers Audio

Les fichiers WAV sont à priori tous supportés quel que soit leur format, mono-canal ou multi-canal. Une fusion des voix sera réalisée en cas de fichier multi-canal. Il faut préciser l'un des quatre formats parmi SMP, SM2, SM4 et DMA pour réaliser l'import du wav (sinon le fichier sera importé tel quel). La fréquence d'échantillonnage n'est pas prise en compte.

- Le format SMP contient une valeur par octet.
- Le format SM2 contient deux valeurs sur un seul octet, le premier échantillon correspond aux bits de poids fort.
- Le format SM4 contient quatre valeurs sur un seul octet, le premier échantillon est encodé dans les bits de poids fort. Le système retenu est le suivant. Les valeurs codées sont les valeurs 0,13,14,15. Ainsi on peut coder sur seulement 2 bits avec toute l'amplitude 4 bits du PSG. Il suffit de copier les deux bits en position 2 et 3. Le zéro reste zéro de facto.
- Avec le format DMA, chaque échantillon sera converti en une liste de commandes DMA. Le sample devra au préalable avoir été converti à 15600Hz.

```
ORG #4000
INCBIN 'sound.wav',SMP
INCBIN 'sound.wav',DMA
```

6.2 Compression

6.2.1 Portions compressées

LZ48 | LZ49 | LZ4 | LZX7 | LZEX0 ... LZCLOSE

Ouvrir un segment de code compressé en LZ48, LZ49, LZ4, ZX7 ou Exomizer. Une section compressée se ferme avec LZCLOSE. Le code produit sera compressé après assemblage et l'ensemble du code situé après la zone sera relogé. Il n'est pas possible d'appeler un label situé après un segment compressé depuis le code compressé pour des raisons évidentes dues aux aléas de la compression. Une erreur s'affichera expliquant pourquoi. Le code ou les données d'une zone LZ ne peut excéder l'espace d'adressage de 64Ko. Enfin, il n'est pas possible d'imbriquer les segments compressés.

```
org #1000
ld hl,zecrunch
ld de,#8000
call decrunch
call #8000
jp next ; label next after crunched zone will be relocated
```

```
zecrunch
lz48 ; -- this section will be crunched
org #8000,$
nop
nop
nop
ret
lzclose ; -- end of crunched section

next
ret
```

6.2.2 Inclusion de fichiers compressés

INCL48, INCL49, INCLZ4, INCZX7, INCEX0 'file to read'

Lire un fichier binaire, le compresser en LZ48, LZ49, LZ4, Exomizer ou ZX7 et l'injecter directement dans le code.

7 Directives et options spécifique à l'Amstrad CPC

RASM intègre des options et des directives qui ne concernent que l'architecture spécifique de l'Amstrad CPC, comme l'accès aux banques mémoire, et de ses émulateurs (export DSK, snapshots). Il offre en outre le support du format cartouche.

7.1 Manipulation des Banques

Les banques mémoires du 6128 peuvent être manipulées en envoyant des données sur le port #7F (le composant gate array), ce qui permet de les échanger par page de 16K. Pour simplifier l'écriture du code, RASM introduit des préfixes qui permettent de récupérer non pas l'adresse d'un label, mais la valeur utilisée pour sélectionner les banques.

7.1.1 Préfixe BANK

Le préfixe {BANK} devant un label (exemple : {BANK}monlabel) permet de récupérer la valeur de la banque mémoire dans laquelle est déclaré le label, plutôt que l'adresse du label. Exemple d'utilisation du tag :

```
BANK 0
ld a,{bank}mysub ; sera assemblé comme LD A,1
call connect_bank
jp mysub

BANK 1
defb 'hello'
mysub
jr $
```

7.1.2 Préfixe PAGE

Le préfixe {PAGE} devant une référence à un label (exemple : {PAGE}monlabel) permet de récupérer la valeur de type Gate array de la BANK dans laquelle est déclaré le label, plutôt que l'adresse du label. Par exemple pour un label situé dans la BANK 5 la valeur retournée sera #7FC5. Si vous utilisez l'adressage mémoire par 64K avec la directive BANKSET, alors la valeur gate array sera déduite du set de 64K ainsi que de l'adresse du label (2 bits de poids fort). Exemple d'utilisation du tag :

```
BANK 0
ld bc,{PAGE}mysub ; sera assemblé comme LD BC,#7FC5
out (c),a
jp mysub

BANK 5
defb 'hello'
mysub
jr $
```

7.1.3 Préfixe PAGESET

Le préfixe {PAGESET} devant un label (exemple : {PAGESET}monlabel) permet de récupérer la valeur de type "Gate Array" du BANKSET dans lequel est déclaré le label, plutôt que l'adresse du label. Par exemple pour un label situé dans la BANK 5 la valeur retournée sera #7FC2

```
BANK 0
  ld a,lo({pageset}mysub) ; sera assemblé comme LD A,#C2
  ld b,#7F
  out (c),a ; Tout la RAM est basculée, le code est suppoosé
  jp mysub ; stocké en ROM ou a un endroit approprié

BANK 5
  defb 'hello'
mysub
jr $
```

7.2 Export de fichiers DSK et entêtes AMSDOS

7.2.1 Entête AMSDOS

L'utilisation de cette directive ajoute un entête Amsdos au fichier binaire produit automatiquement par Rasm. Cet entête est sans effet sur la directive **SAVE** que nous allons voir au prochain paragraphe (elle dispose de sa propre option pour ajouter cet entête à la demande).

7.2.2 Directive SAVE

```
SAVE 'filename',<address>,<size>[,AMSDOS|DSK[, 'dsk filename'[,<side>]]]
```

Enregistre un fichier binaire à partir de la mémoire adresse jusqu'à adresse+taille de l'espace mémoire en cours. Bien que l'instruction **SAVE** puisse être déclarée à n'importe quel moment, les enregistrements de fichier sont toujours réalisés en fin d'assemblage si et seulement si il n'y a pas eu d'erreur. Il n'est donc pas possible d'enregistrer des états intermédiaires d'assemblage.

Lorsqu'on enregistre sur une image de disquette (DSK), le nom du fichier binaire est automatiquement tronqué et mis en majuscule si il n'est pas conforme aux limitations de l'AMSDOS. Notez aussi que si le fichier DSK existe déjà, RASM va vérifier si le fichier binaire généré existe déjà dans le DSK. Si il existe, il refusera de mettre à jour le DSK, sauf si on passe l'option **-eo**. Si le DSK n'existe pas, il sera automatiquement créé.

Exemples :

```
;Sauve un fichier binaire
SAVE 'myfile.bin',start,size

;Sauve un binaire avec un header AMSDOS
SAVE 'myfile.bin',start,size,AMSDOS

;Sauve un binaire AMSDOS dans un fichier DSK
SAVE 'myfile.bin',start,size,DSK,'fichierdsk.dsk'
```

Combiné avec RUN :

```
ORG #9000

start:
  call #bb06
  ret
end:

RUN start
SAVE 'main.bin',start,end-start,DSK,'main.dsk'
```

7.3 Cartouches et Snapshots

En plus du format DSK, RASM permet l'exportation de fichiers au format cartridge et snapshot. Ils peuvent être utilisés par certains émulateurs (Winape, Ace, etc.). La méthode pour produire les deux formats est très similaire. En début de programme, il faut utiliser les directives `BUILDCPR` ou `BUILDSNA`, `BANK` et `RUN`. C'est la directive `BANK` qui va provoquer la génération d'une cartouche ou d'un snapshot.

Génération d'une Cartouche Pour générer une cartouche, il suffit que le source débute par les deux lignes suivantes :

```
| BUILDCPR  
| BANK 0
```

La directive `BUILDCPR` n'a pas vraiment d'utilité, car par défaut, en cas d'utilisation de la directive `BANK`, c'est une cartouche qui est générée. Mais pour des raisons de lisibilité débiter le source par cette directive permet d'enlever toute ambiguïté. Il n'est pas nécessaire de préciser le point d'entrée, car il est forcément à l'adresse 0.

Génération d'un Snapshot La génération d'un snapshot nécessite aussi quelques lignes, très similaires à la cartouche. Ici avec un programme dont le point d'entrée à l'adresse `#A000` :

```
| BUILDSNA  
| BANK 0  
| RUN #A000
```

Il est aussi possible d'utiliser la directive `BANKSET` pour générer un fichier snapshot.

7.3.1 BUILDSNA

`BUILDSNA [V2]`

L'usage de la directive `BUILDSNA` indique à Rasm que l'on veut générer un snapshot et non une cartouche. De plus, à la différence d'une cartouche, il faut préciser le point d'entrée (0 n'étant pas valide). Le snapshot généré par Rasm inclut un écran de taille classique (le même que sous Basic), les encres sont aux valeurs par défaut du Basic (non clignotantes). Les 3 voies audio sont désactivées, les roms désactivées et le mode d'interruption est 1. Par défaut le snapshot est initialisé avec un 6128 CRTC 0 mais il est possible avec les directives `SETCRTC` et `SETCPC` de choisir un autre type de CPC et de CRTC.

7.3.2 BANK

```
BANK [ROM page number]  
BANK [RAM page number]
```

Sélectionner un emplacement ROM (cartouche) ou RAM (snapshot). L'usage de cette instruction active par défaut l'écriture de la cartouche en fin d'assemblage. Les valeurs possibles vont de 0 à 31. En mode snapshot on peut aussi utiliser cette directive, cette fois avec des valeurs de 0 à 35 (64K de base + 512K d'extension mémoire).

Sans paramètre, la directive ouvre un nouvel espace mémoire de travail

```
| BUILDSNA ; recommended usage when using snapshot is to set it first  
| BANKSET 0 ; assembling in first 64K  
| ORG #1000  
| RUN #1000 ; entry point is set to #1000
```

```
ld b,#7F
ld a,{page}mydata ; get gate array value for paging memory
out (c),a
ld a,(mydata)
jr $

BANK 6 ; choose 3th bank of 2nd 64K set
nop
mydata defb #DD

bank
; bank used without parameter, this is a temporary memory space
; that won't be saved in the snapshot

pouet
repeat 10
    cpi
    rend
camion
SAVE"another",pouet,camion-pouet
```

Il existe une option de compatibilité pour la création de snapshot version 2 : Certains émulateurs ou cartes hardware ne gèrent pas encore le format v3. Pour rétrograder les snapshots en format v2 (128K maximum, non compressés), il suffit d'ajouter le paramètre V2 après la directive ou de passer le paramètre -v2 au lancement de RASM.

7.3.3 RUN

RUN <address>[,<gate array configuration>]

Cette directive n'est prise en compte que si on génère un snapshot. Alors l'adresse de démarrage du code sera injectée dans le snapshot. En option on peut spécifier la configuration du gate array, typiquement pour exécuter un programme depuis les 64K de mémoire étendue.

7.4 Directives spécifiques aux snapshots

7.4.1 BANKSET

BANKSET <64K bloc number>

Sert à sélectionner un emplacement mémoire pour les snapshots, en groupant les pages 4 à 4. Le format snapshot v3 supportant au maximum une extension de 512K, il y a 9 sets mémoire indexés de 0 à 8. Il est possible d'utiliser BANK et BANKSET en même temps mais il est impératif de ne pas sélectionner la même page à la fois avec BANK et BANKSET. Un contrôle déclenchera une erreur si vous tentez de le faire.

L'appel de cette directive force automatiquement la génération de snapshot.

7.4.2 SETCPC

SETCPC <model>

Choisir le modèle de CPC quand on enregistre un snapshot. Les valeurs autorisées sont :

0 : CPC 464
1 : CPC 664
2 : CPC 6128
4 : 464 Plus
5 : 6128 Plus
6 : GX-4000

7.4.3 SETCRTC

SETCRTC <CRTC model>

Choisir le modèle de CRTC quand on enregistre un snapshot. Les valeurs autorisées vont de 0 à 4. Pour rappel, les CPC ont des CRTC 0,1,2 ou 4 et les Plus ou GX-4000 ont tous le CRTC 3.

7.4.4 BREAKPOINT

BREAKPOINT [<address>]
[@]BRKlabel

Ajoute un point d'arrêt (ce n'est pas une instruction qui est assemblée) avec pour adresse de break l'adresse de l'instruction suivante ou celle du paramètre optionnel. Les points d'arrêt peuvent être exportés sous forme de fichier brut ou dans les snapshots (compatible avec les émulateurs ACE et Winape) avec l'option -sb. Note : Tout label qui commence par le préfixe BRK ou @BRK génère à la fois un label et un point d'arrêt.

7.4.5 Options d'export

Voici les options qui peuvent être passées en ligne de commande pour spécifier les noms des fichiers exportés et activer l'export des symboles et breakpoints pour le format SNA :

- -oc <nom du fichier cartouche> : spécifie le nom complet du fichier cartouche.
- -oi <fichier snapshot > spécifie le nom complet du fichier snapshot
- -v2 : Crée un snapshot version 2 (export par défaut en version 3)
- -ss : Exporte les symboles dans le fichier snapshot (uniquement compatible avec ACE), uniquement avec la version 3+
- -ok <breakpoint filename> : spécifie le nom complet du fichier breakpoint.
- -eb : Exporte les breakpoints dans un fichier texte
- -sb : Exporte les points d'arrêt (breakpoints) dans le fichier snapshot (compatible avec les formats Winape et ACE), uniquement avec la version 3+

7.5 Gestion des couleurs du CPC+

7.5.1 GET_R, GET_G, GET_B

GET_R <16 bits RGB value>
GET_G <16 bits RGB value>
GET_B <16 bits RGB value>

Permet, dans une expression, de récupérer l'une des 3 composantes (4 bits) R,G,B d'une couleur au format utilisé par l'ASIC

7.5.2 SET_R, SET_G, SET_B

```
SET_R <4 bits value>  
SET_G <4 bits value>  
SET_B <4 bits value>
```

Renvoie une valeur 16 bit compatible avec le format de l'ASIC, dont la composante R,G ou B est fixée selon le parametre passé.

```
| dw (SET_R 4) | (SET_G 15) | (SET_B 0) ; Defines RGB Color (4,15,0)
```

ou encore

```
| macro drgb dr,db,dg  
| dw SET_R dr | SET_G dg | SET_B db  
| mend
```

7.6 Directives obsolètes

7.6.1 NOCODE

```
NOCODE  
...  
CODE
```

Cette directive sert à désactiver la génération de code pour une portion de code. Tout se passe comme si le code était généré, mais a la sortie de la zone définie par CODE, la suite du code sera généré la ou la section avait commencé. Cette directive servait à définir des structures de données, elle est maintenant obsolète pour cet usage. Elle peut cependant permettre de faire des calculs basés sur du code, de logger le résultat de ces calculs dans des variables, et pourtant ne rien produire en sortie.

7.6.2 WRITE DIRECT

```
WRITE DIRECT <lower rom>[,<higher rom>[,<RAM gate array>]]
```

Cette directive est présente pour compatibilité avec Winape, son usage est déconseillé. Utilisez de préférence les directives BANK ou BANKSET. En spécifiant une rom basse (entre 0 et 7) ou une rom haute (entre 0 et 31), cette directive a le même effet que la directive BANK. En spécifiant uniquement l'adresse RAM (n'importe quelle valeur) et en désactivant les numéros de rom avec la valeur -1, on crée à chaque appel un nouvel espace mémoire. On peut ainsi assembler plusieurs code au même emplacement mémoire, mais dans un espace différent. Cet usage est équivalent à la directive BANK sans paramètre.

7.6.3 LIST, NOLIST, LET

Ces directives sont ignorées, elles n'existent que pour la compatibilité avec Maxam et Winape.

8 Directives et options spécifique au ZX

RASM intègre aussi quelques options et directives qui ne concernent que l'architecture spécifique du ZX.

8.1 Selection des Banques

8.1.1 Directive BUILDZX

BUILDZX <bank>

Cette directive sert a selection l'une des 8 banques (0..7)

8.2 Options spécifiques

l'option **-sx** permet de réaliser un export pour certaines émulateurs ZX, en précisant la banque où le symbole est logé ((<bank>:<adresse>))

9 Compilation

Cette section décrit la procédure pour compiler RASM sur différentes plateformes, le code source C étant disponible. De plus, il est possible d'intégrer RASM dans vos propres applications, ce qui sera vu à la prochaine suivante.

9.1 Compiler RASM

Pour commencer, voici les commandes à exécuter pour quelques plateformes courantes.

9.1.1 Linux

```
cc rasm_vXXX.c -O2 -lm -lrt -march=native
mv a.out rasm
strip rasm
```

9.1.2 Windows (Visual Studio)

```
cl.exe rasm_vXXX.c -O2 -Ob3
```

9.1.3 Dos/Windows 32 (Watcom)

```
wcl386.exe rasm_vXXX.c -6r -6s -fp6 -d0 -k4000000 -ox
```

9.1.4 MacOS

```
cc rasm_vXXX.c -O2 -lm -march=native
```

9.1.5 MorphOS

```
ppc-morphos-gcc-5 -O2 -c -o rasm rasm_vXXX.c
strip rasm
```

9.2 Intégrer RASM

Pour intégrer RASM dans votre propre application C/C++, il faut procéder en trois étapes :

- Compiler RASM sous forme de binaire (obj), avec le symbole `INTEGRATED_ASSEMBLY` défini
- Inclure `'rasm.h'` et utiliser l'une des deux fonctions pour compiler du code, ainsi que les structures de données définies dans ce header, si l'on souhaite récupérer des informations sur le processus d'assemblage
- Compiler votre programme en linkant le binaire produit à l'étape 1.

Dans notre premier exemple, nous allons utiliser la fonction `RasmAssemble`, qui renvoie, en plus du code assemblé, un code d'erreur, 0 si tout s'est bien passé, -1 sinon. La fonction `RasmAssembleInfos` est identique, à ceci près qu'elle renvoie une structure de donnée contenant des informations sur le processus d'assemblage : les éventuelles erreurs, et la valeur associée à chacun des symboles.

```
#include "rasm.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Program to assemble
const char* prog="org #9000 \n\
```

```
    ld b,10 \n\
lp: djnz lp \n\
    ret \n";

int main(void) {
    printf("%s\n", prog);

    unsigned char *buf = NULL;
    int asmsize = 0;
    int res = RasmAssemble(prog, strlen(prog), &buf, &asmsize);

    printf("Result=%d, Generated code size=%d\n",res,asmsize);

    if (res==0)
    {
        for (int i=0; i<asmsize; i++)
        {
            printf("%02X ", buf[i]);
            if ((i&15)==15) printf("\n");
        }
        printf("\n");
    }
    else
    {
        printf("Failure!\n");
    }

    if (buf)
        free(buf);

    return 0;
}
```

Pour compiler ce fichier (embed.cpp) sous linux avec gcc :

```
gcc -D INTEGRATED_ASSEMBLY rasm_v0111.c -c -o rasm_embedded.obj
gcc embed.cpp rasm_embedded.obj -lm
```

Le résultat de l'exécution est le suivant :

```
$ ./a.out
org #9000
ld b,10
lp: djnz lp
ret
Result=0, Generated code size=5
06 0A 10 FE C9
```

9.2.1 Récupération des erreurs et symboles

Dans l'exemple précédent, on a vu que le résultat de la fonction `RasmAssemble` était -1 en cas d'erreur. Si l'on souhaite d'avantages d'informations, il faudra appeler `RasmAssembleInfos` qui renvoie une structure contenant la liste des erreurs ainsi que la liste des symboles avec leur valeur.

A Coloration syntaxique

A.1 Coloration syntaxique dans VIM

Si vous avez déjà le fichier de coloration syntaxique Z80 il suffit d'ajouter les lignes suivantes au fichier `.vim/syntax/z80.vim` Ou vous pouvez télécharger le fichier complet de syntaxe `z80.vim`

```
" rasm/winape directives
syn keyword z80PreProc charset bank write save include incbin incl48 incl49
syn keyword z80PreProc macro mend switch case break while wend repeat until
syn keyword z80PreProc buildcpr amsdos lz48 lz49 lzclos protect
syn keyword z80PreProc direct brk let print stop nolist str
syn keyword z80PreProc defr dr defi undef
syn keyword z80PreProc bankset page pageset sizeof endm struct endstruct ends
syn keyword z80PreProc incexo lzexo lzx7 inczx7 buildsna setcrtc setcpc assert print
syn keyword z80Reg lix liy hix hiy
```

B Opcodes Z80

B.1 Instructions principales

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nop	ld bc,**	ld (bc),a	inc bc	inc b	dec b	ld b,*	rlca	ex af,af'	add hl,bc	ld a,(bc)	dec bc	inc c	dec c	ld c,*	rrca
1	djnz *	ld de,**	ld (de),a	inc de	inc d	dec d	ld d,*	rla	jr *	add hl,de	ld a,(de)	dec de	inc e	dec e	ld e,*	rra
2	jr nz,*	ld hl,**	ld (**),hl	inc hl	inc h	dec h	ld h,*	daa	jr z,*	add hl,hl	ld hl,(**)	dec hl	inc l	dec l	ld l,*	cpl
3	jr nc,*	ld sp,**	ld (**),a	inc sp	inc (hl)	dec (hl)	ld (hl),*	scf	jr c,*	add hl,sp	ld a,(**)	dec sp	inc a	dec a	ld a,*	ccf
4	ld b,b	ld b,c	ld b,d	ld b,e	ld b,h	ld b,l	ld b,(hl)	ld b,a	ld c,b	ld c,c	ld c,d	ld c,e	ld c,h	ld c,l	ld c,(hl)	ld c,a
5	ld d,b	ld d,c	ld d,d	ld d,e	ld d,h	ld d,l	ld d,(hl)	ld d,a	ld e,b	ld e,c	ld e,d	ld e,e	ld e,h	ld e,l	ld e,(hl)	ld e,a
6	ld h,b	ld h,c	ld h,d	ld h,e	ld h,h	ld h,l	ld h,(hl)	ld h,a	ld l,b	ld l,c	ld l,d	ld l,e	ld l,h	ld l,l	ld l,(hl)	ld l,a
7	ld (hl),b	ld (hl),c	ld (hl),d	ld (hl),e	ld (hl),h	ld (hl),l	halt	ld (hl),a	ld a,b	ld a,c	ld a,d	ld a,e	ld a,h	ld a,l	ld a,(hl)	ld a,a
8	add a,b	add a,c	add a,d	add a,e	add a,h	add a,l	add a,(hl)	add a,a	adc a,b	adc a,c	adc a,d	adc a,e	adc a,h	adc a,l	adc a,(hl)	adc a,a
9	sub b	sub c	sub d	sub e	sub h	sub l	sub (hl)	sub a	sbc a,b	sbc a,c	sbc a,d	sbc a,e	sbc a,h	sbc a,l	sbc a,(hl)	sbc a,a
A	and b	and c	and d	and e	and h	and l	and (hl)	and a	xor b	xor c	xor d	xor e	xor h	xor l	xor (hl)	xor a
B	or b	or c	or d	or e	or h	or l	or (hl)	or a	cp b	cp c	cp d	cp e	cp h	cp l	cp (hl)	cp a
C	ret nz	pop bc	jp nz,**	jp **	call nz,**	push bc	add a,*	rst #00	ret z	ret	jp z,**	Préfixe CB	call z,**	call **	adc a,*	rst #08
D	ret nc	pop de	jp nc,**	out (*),a	call nc,**	push de	sub *	rst #10	ret c	exx	jp c,**	in a,(*)	call c,**	Préfixe DD	sbc a,*	rst #18
E	ret po	pop hl	jp po,**	ex (sp),hl	call po,**	push hl	and *	rst #20	ret pe	jp (hl)	jp pe,**	ex de,hl	call pe,**	Préfixe ED	xor *	rst #28
F	ret p	pop af	jp p,**	di	call p,**	push af	or *	rst #30	ret m	ld sp,hl	jp m,**	ei	call m,**	Préfixe FD	cp *	rst #38

B.2 Instructions étendues (ED)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4	in b,(c)	out (c),b	sbc hl,bc	ld (**),bc	neg	retn	im 0	ld i,a	in c,(c)	out (c),c	adc hl,bc	ld bc,(**)	neg	reti	im 0/1	ld r,a
5	in d,(c)	out (c),d	sbc hl,de	ld (**),de	neg	retn	im 1	ld a,i	in e,(c)	out (c),e	adc hl,de	ld de,(**)	neg	retn	im 2	ld a,r
6	in h,(c)	out (c),h	sbc hl,hl	ld (**),hl	neg	retn	im 0	rrd	in l,(c)	out (c),l	adc hl,hl	ld hl,(**)	neg	retn	im 0/1	rld
7	in (c)	out (c),0	sbc hl,sp	ld (**),sp	neg	retn	im 1		in a,(c)	out (c),a	adc hl,sp	ld sp,(**)	neg	retn	im 2	
A	ldi	cpi	ini	outi					ldd	cpd	ind	outd				
B	ldir	cpir	inir	otir					lddr	cpdr	indr	otdr				

B.3 Instruction de manipulation de bits (CB)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	rlc b	rlc c	rlc d	rlc e	rlc h	rlc l	rlc (hl)	rlc a	rrc b	rrc c	rrc d	rrc e	rrc h	rrc l	rrc (hl)	rrc a
1	rl b	rl c	rl d	rl e	rl h	rl l	rl (hl)	rl a	rr b	rr c	rr d	rr e	rr h	rr l	rr (hl)	rr a
2	sla b	sla c	sla d	sla e	sla h	sla l	sla (hl)	sla a	sra b	sra c	sra d	sra e	sra h	sra l	sra (hl)	sra a
3	sll b	sll c	sll d	sll e	sll h	sll l	sll (hl)	sll a	srl b	srl c	srl d	srl e	srl h	srl l	srl (hl)	srl a
4	bit 0,b	bit 0,c	bit 0,d	bit 0,e	bit 0,h	bit 0,l	bit 0,(hl)	bit 0,a	bit 1,b	bit 1,c	bit 1,d	bit 1,e	bit 1,h	bit 1,l	bit 1,(hl)	bit 1,a
5	bit 2,b	bit 2,c	bit 2,d	bit 2,e	bit 2,h	bit 2,l	bit 2,(hl)	bit 2,a	bit 3,b	bit 3,c	bit 3,d	bit 3,e	bit 3,h	bit 3,l	bit 3,(hl)	bit 3,a
6	bit 4,b	bit 4,c	bit 4,d	bit 4,e	bit 4,h	bit 4,l	bit 4,(hl)	bit 4,a	bit 5,b	bit 5,c	bit 5,d	bit 5,e	bit 5,h	bit 5,l	bit 5,(hl)	bit 5,a
7	bit 6,b	bit 6,c	bit 6,d	bit 6,e	bit 6,h	bit 6,l	bit 6,(hl)	bit 6,a	bit 7,b	bit 7,c	bit 7,d	bit 7,e	bit 7,h	bit 7,l	bit 7,(hl)	bit 7,a
8	res 0,b	res 0,c	res 0,d	res 0,e	res 0,h	res 0,l	res 0,(hl)	res 0,a	res 1,b	res 1,c	res 1,d	res 1,e	res 1,h	res 1,l	res 1,(hl)	res 1,a
9	res 2,b	res 2,c	res 2,d	res 2,e	res 2,h	res 2,l	res 2,(hl)	res 2,a	res 3,b	res 3,c	res 3,d	res 3,e	res 3,h	res 3,l	res 3,(hl)	res 3,a
A	res 4,b	res 4,c	res 4,d	res 4,e	res 4,h	res 4,l	res 4,(hl)	res 4,a	res 5,b	res 5,c	res 5,d	res 5,e	res 5,h	res 5,l	res 5,(hl)	res 5,a
B	res 6,b	res 6,c	res 6,d	res 6,e	res 6,h	res 6,l	res 6,(hl)	res 6,a	res 7,b	res 7,c	res 7,d	res 7,e	res 7,h	res 7,l	res 7,(hl)	res 7,a
C	set 0,b	set 0,c	set 0,d	set 0,e	set 0,h	set 0,l	set 0,(hl)	set 0,a	set 1,b	set 1,c	set 1,d	set 1,e	set 1,h	set 1,l	set 1,(hl)	set 1,a
D	set 2,b	set 2,c	set 2,d	set 2,e	set 2,h	set 2,l	set 2,(hl)	set 2,a	set 3,b	set 3,c	set 3,d	set 3,e	set 3,h	set 3,l	set 3,(hl)	set 3,a
E	set 4,b	set 4,c	set 4,d	set 4,e	set 4,h	set 4,l	set 4,(hl)	set 4,a	set 5,b	set 5,c	set 5,d	set 5,e	set 5,h	set 5,l	set 5,(hl)	set 5,a
F	set 6,b	set 6,c	set 6,d	set 6,e	set 6,h	set 6,l	set 6,(hl)	set 6,a	set 7,b	set 7,c	set 7,d	set 7,e	set 7,h	set 7,l	set 7,(hl)	set 7,a

B.4 Instructions IX (DD)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										add ix,bc						
1										add ix,de						
2		ld ix,**	ld (**),ix	inc ix	inc ixh	dec ixh	ld ixh,*			add ix,ix	ld ix,(**)	dec ix	inc ixl	dec ixl	ld ixl,*	
3					inc (ix++)	dec (ix++)	ld (ix++),*			add ix,sp						
4					ld b,ixh	ld b,ixl	ld b,(ix++)						ld c,ixh	ld c,ixl	ld c,(ix++)	
5					ld d,ixh	ld d,ixl	ld d,(ix++)						ld e,ixh	ld e,ixl	ld e,(ix++)	
6	ld ixh,b	ld ixh,c	ld ixh,d	ld ixh,e	ld ixh,ixh	ld ixh,ixl	ld h,(ix++)	ld ixh,a	ld ixl,b	ld ixl,c	ld ixl,d	ld ixl,e	ld ixl,ixh	ld ixl,ixl	ld l,(ix++)	ld ixl,a
7	ld (ix++),b	ld (ix++),c	ld (ix++),d	ld (ix++),e	ld (ix++),h	ld (ix++),l		ld (ix++),a					ld a,ixh	ld a,ixl	ld a,(ix++)	
8					add a,ixh	add a,ixl	add a,(ix++)						adc a,ixh	adc a,ixl	adc a,(ix++)	
9					sub ixh	sub ixl	sub (ix++)						sbc a,ixh	sbc a,ixl	sbc a,(ix++)	
A					and ixh	and ixl	and (ix++)						xor ixh	xor ixl	xor (ix++)	
B					or ixh	or ixl	or (ix++)						cp ixh	cp ixl	cp (ix++)	
C												Préfixe DDCB				
D																
E		pop ix		ex (sp),ix		push ix				jp (ix)						
F										ld sp,ix						

B.5 Instruction de manipulations de bits IX (DDCB)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	rlc (ix**),b	rlc (ix**),c	rlc (ix**),d	rlc (ix**),e	rlc (ix**),h	rlc (ix**),l	rlc (ix**),	rlc (ix**),a	rrc (ix**),b	rrc (ix**),c	rrc (ix**),d	rrc (ix**),e	rrc (ix**),h	rrc (ix**),l	rrc (ix**),	rrc (ix**),a
1	rl (ix**),b	rl (ix**),c	rl (ix**),d	rl (ix**),e	rl (ix**),h	rl (ix**),l	rl (ix**),	rl (ix**),a	rr (ix**),b	rr (ix**),c	rr (ix**),d	rr (ix**),e	rr (ix**),h	rr (ix**),l	rr (ix**),	rr (ix**),a
2	sla (ix**),b	sla (ix**),c	sla (ix**),d	sla (ix**),e	sla (ix**),h	sla (ix**),l	sla (ix**),	sla (ix**),a	sra (ix**),b	sra (ix**),c	sra (ix**),d	sra (ix**),e	sra (ix**),h	sra (ix**),l	sra (ix**),	sra (ix**),a
3	sll (ix**),b	sll (ix**),c	sll (ix**),d	sll (ix**),e	sll (ix**),h	sll (ix**),l	sll (ix**),	sll (ix**),a	srl (ix**),b	srl (ix**),c	srl (ix**),d	srl (ix**),e	srl (ix**),h	srl (ix**),l	srl (ix**),	srl (ix**),a
4	bit 0, (ix**),	bit 0, (ix**),	bit 0, (ix**),	bit 0, (ix**),	bit 0, (ix**),	bit 0, (ix**),	bit 0, (ix**),	bit 0, (ix**),	bit 1, (ix**),	bit 1, (ix**),	bit 1, (ix**),	bit 1, (ix**),	bit 1, (ix**),	bit 1, (ix**),	bit 1, (ix**),	bit 1, (ix**),
5	bit 2, (ix**),	bit 2, (ix**),	bit 2, (ix**),	bit 2, (ix**),	bit 2, (ix**),	bit 2, (ix**),	bit 2, (ix**),	bit 2, (ix**),	bit 3, (ix**),	bit 3, (ix**),	bit 3, (ix**),	bit 3, (ix**),	bit 3, (ix**),	bit 3, (ix**),	bit 3, (ix**),	bit 3, (ix**),
6	bit 4, (ix**),	bit 4, (ix**),	bit 4, (ix**),	bit 4, (ix**),	bit 4, (ix**),	bit 4, (ix**),	bit 4, (ix**),	bit 4, (ix**),	bit 5, (ix**),	bit 5, (ix**),	bit 5, (ix**),	bit 5, (ix**),	bit 5, (ix**),	bit 5, (ix**),	bit 5, (ix**),	bit 5, (ix**),
7	bit 6, (ix**),	bit 6, (ix**),	bit 6, (ix**),	bit 6, (ix**),	bit 6, (ix**),	bit 6, (ix**),	bit 6, (ix**),	bit 6, (ix**),	bit 7, (ix**),	bit 7, (ix**),	bit 7, (ix**),	bit 7, (ix**),	bit 7, (ix**),	bit 7, (ix**),	bit 7, (ix**),	bit 7, (ix**),
8	res 0, (ix**),b	res 0, (ix**),c	res 0, (ix**),d	res 0, (ix**),e	res 0, (ix**),h	res 0, (ix**),l	res 0, (ix**),	res 0, (ix**),a	res 1, (ix**),b	res 1, (ix**),c	res 1, (ix**),d	res 1, (ix**),e	res 1, (ix**),h	res 1, (ix**),l	res 1, (ix**),	res 1, (ix**),a
9	res 2, (ix**),b	res 2, (ix**),c	res 2, (ix**),d	res 2, (ix**),e	res 2, (ix**),h	res 2, (ix**),l	res 2, (ix**),	res 2, (ix**),a	res 3, (ix**),b	res 3, (ix**),c	res 3, (ix**),d	res 3, (ix**),e	res 3, (ix**),h	res 3, (ix**),l	res 3, (ix**),	res 3, (ix**),a
A	res 4, (ix**),b	res 4, (ix**),c	res 4, (ix**),d	res 4, (ix**),e	res 4, (ix**),h	res 4, (ix**),l	res 4, (ix**),	res 4, (ix**),a	res 5, (ix**),b	res 5, (ix**),c	res 5, (ix**),d	res 5, (ix**),e	res 5, (ix**),h	res 5, (ix**),l	res 5, (ix**),	res 5, (ix**),a
B	res 6, (ix**),b	res 6, (ix**),c	res 6, (ix**),d	res 6, (ix**),e	res 6, (ix**),h	res 6, (ix**),l	res 6, (ix**),	res 6, (ix**),a	res 7, (ix**),b	res 7, (ix**),c	res 7, (ix**),d	res 7, (ix**),e	res 7, (ix**),h	res 7, (ix**),l	res 7, (ix**),	res 7, (ix**),a
C	set 0, (ix**),b	set 0, (ix**),c	set 0, (ix**),d	set 0, (ix**),e	set 0, (ix**),h	set 0, (ix**),l	set 0, (ix**),	set 0, (ix**),a	set 1, (ix**),b	set 1, (ix**),c	set 1, (ix**),d	set 1, (ix**),e	set 1, (ix**),h	set 1, (ix**),l	set 1, (ix**),	set 1, (ix**),a
D	set 2, (ix**),b	set 2, (ix**),c	set 2, (ix**),d	set 2, (ix**),e	set 2, (ix**),h	set 2, (ix**),l	set 2, (ix**),	set 2, (ix**),a	set 3, (ix**),b	set 3, (ix**),c	set 3, (ix**),d	set 3, (ix**),e	set 3, (ix**),h	set 3, (ix**),l	set 3, (ix**),	set 3, (ix**),a
E	set 4, (ix**),b	set 4, (ix**),c	set 4, (ix**),d	set 4, (ix**),e	set 4, (ix**),h	set 4, (ix**),l	set 4, (ix**),	set 4, (ix**),a	set 5, (ix**),b	set 5, (ix**),c	set 5, (ix**),d	set 5, (ix**),e	set 5, (ix**),h	set 5, (ix**),l	set 5, (ix**),	set 5, (ix**),a
F	set 6, (ix**),b	set 6, (ix**),c	set 6, (ix**),d	set 6, (ix**),e	set 6, (ix**),h	set 6, (ix**),l	set 6, (ix**),	set 6, (ix**),a	set 7, (ix**),b	set 7, (ix**),c	set 7, (ix**),d	set 7, (ix**),e	set 7, (ix**),h	set 7, (ix**),l	set 7, (ix**),	set 7, (ix**),a

B.6 Instructions IY (FD)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										add iy,bc						
1										add iy,de						
2		ld iy,**	ld (**),iy	inc iy	inc iyh	dec iyh	ld iyh,*			add iy,iy	ld iy,(**)	dec iy	inc iyl	dec iyl	ld iyl,*	
3					inc (iy++)	dec (iy++)	ld (iy++),*			add iy,sp						
4					ld b,iyh	ld b,iyl	ld b, (iy++)						ld c,iyh	ld c,iyl	ld c, (iy++)	
5					ld d,iyh	ld d,iyl	ld d, (iy++)						ld e,iyh	ld e,iyl	ld e, (iy++)	
6	ld iyh,b	ld iyh,c	ld iyh,d	ld iyh,e	ld iyh,iyh	ld iyh,iyl	ld h, (iy++)	ld iyh,a	ld iyl,b	ld iyl,c	ld iyl,d	ld iyl,e	ld iyl,iyh	ld iyl,iyl	ld l, (iy++)	ld iyl,a
7	ld (iy++),b	ld (iy++),c	ld (iy++),d	ld (iy++),e	ld (iy++),h	ld (iy++),l		ld (iy++),a					ld a,iyh	ld a,iyl	ld a, (iy++)	
8					add a,iyh	add a,iyl	add a, (iy++)						adc a,iyh	adc a,iyl	adc a, (iy++)	
9					sub iyh	sub iyl	sub (iy++)						sbc a,iyh	sbc a,iyl	sbc a, (iy++)	
A					and iyh	and iyl	and (iy++)						xor iyh	xor iyl	xor (iy++)	
B					or iyh	or iyl	or (iy++)						cp iyh	cp iyl	cp (iy++)	
C												Préfixe FDCB				
D																
E		pop iy		ex (sp),iy		push iy				jp (iy)						
F										ld sp,iy						

B.7 Instruction de manipulation de bits IY (FDCB)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	rlc (iy++),b	rlc (iy++),c	rlc (iy++),d	rlc (iy++),e	rlc (iy++),h	rlc (iy++),l	rlc (iy++)	rlc (iy++),a	rrc (iy++),b	rrc (iy++),c	rrc (iy++),d	rrc (iy++),e	rrc (iy++),h	rrc (iy++),l	rrc (iy++)	rrc (iy++),a
1	rl (iy++),b	rl (iy++),c	rl (iy++),d	rl (iy++),e	rl (iy++),h	rl (iy++),l	rl (iy++)	rl (iy++),a	rr (iy++),b	rr (iy++),c	rr (iy++),d	rr (iy++),e	rr (iy++),h	rr (iy++),l	rr (iy++)	rr (iy++),a
2	sla (iy++),b	sla (iy++),c	sla (iy++),d	sla (iy++),e	sla (iy++),h	sla (iy++),l	sla (iy++)	sla (iy++),a	sra (iy++),b	sra (iy++),c	sra (iy++),d	sra (iy++),e	sra (iy++),h	sra (iy++),l	sra (iy++)	sra (iy++),a
3	sll (iy++),b	sll (iy++),c	sll (iy++),d	sll (iy++),e	sll (iy++),h	sll (iy++),l	sll (iy++)	sll (iy++),a	srl (iy++),b	srl (iy++),c	srl (iy++),d	srl (iy++),e	srl (iy++),h	srl (iy++),l	srl (iy++)	srl (iy++),a
4	bit 0, (iy++)	bit 0, (iy++)	bit 0, (iy++)	bit 0, (iy++)	bit 0, (iy++)	bit 0, (iy++)	bit 0, (iy++)	bit 0, (iy++)	bit 1, (iy++)	bit 1, (iy++)	bit 1, (iy++)	bit 1, (iy++)	bit 1, (iy++)	bit 1, (iy++)	bit 1, (iy++)	bit 1, (iy++)
5	bit 2, (iy++)	bit 2, (iy++)	bit 2, (iy++)	bit 2, (iy++)	bit 2, (iy++)	bit 2, (iy++)	bit 2, (iy++)	bit 2, (iy++)	bit 3, (iy++)	bit 3, (iy++)	bit 3, (iy++)	bit 3, (iy++)	bit 3, (iy++)	bit 3, (iy++)	bit 3, (iy++)	bit 3, (iy++)
6	bit 4, (iy++)	bit 4, (iy++)	bit 4, (iy++)	bit 4, (iy++)	bit 4, (iy++)	bit 4, (iy++)	bit 4, (iy++)	bit 4, (iy++)	bit 5, (iy++)	bit 5, (iy++)	bit 5, (iy++)	bit 5, (iy++)	bit 5, (iy++)	bit 5, (iy++)	bit 5, (iy++)	bit 5, (iy++)
7	bit 6, (iy++)	bit 6, (iy++)	bit 6, (iy++)	bit 6, (iy++)	bit 6, (iy++)	bit 6, (iy++)	bit 6, (iy++)	bit 6, (iy++)	bit 7, (iy++)	bit 7, (iy++)	bit 7, (iy++)	bit 7, (iy++)	bit 7, (iy++)	bit 7, (iy++)	bit 7, (iy++)	bit 7, (iy++)
8	res 0, (iy++),b	res 0, (iy++),c	res 0, (iy++),d	res 0, (iy++),e	res 0, (iy++),h	res 0, (iy++),l	res 0, (iy++)	res 0, (iy++),a	res 1, (iy++),b	res 1, (iy++),c	res 1, (iy++),d	res 1, (iy++),e	res 1, (iy++),h	res 1, (iy++),l	res 1, (iy++)	res 1, (iy++),a
9	res 2, (iy++),b	res 2, (iy++),c	res 2, (iy++),d	res 2, (iy++),e	res 2, (iy++),h	res 2, (iy++),l	res 2, (iy++)	res 2, (iy++),a	res 3, (iy++),b	res 3, (iy++),c	res 3, (iy++),d	res 3, (iy++),e	res 3, (iy++),h	res 3, (iy++),l	res 3, (iy++)	res 3, (iy++),a
A	res 4, (iy++),b	res 4, (iy++),c	res 4, (iy++),d	res 4, (iy++),e	res 4, (iy++),h	res 4, (iy++),l	res 4, (iy++)	res 4, (iy++),a	res 5, (iy++),b	res 5, (iy++),c	res 5, (iy++),d	res 5, (iy++),e	res 5, (iy++),h	res 5, (iy++),l	res 5, (iy++)	res 5, (iy++),a
B	res 6, (iy++),b	res 6, (iy++),c	res 6, (iy++),d	res 6, (iy++),e	res 6, (iy++),h	res 6, (iy++),l	res 6, (iy++)	res 6, (iy++),a	res 7, (iy++),b	res 7, (iy++),c	res 7, (iy++),d	res 7, (iy++),e	res 7, (iy++),h	res 7, (iy++),l	res 7, (iy++)	res 7, (iy++),a
C	set 0, (iy++),b	set 0, (iy++),c	set 0, (iy++),d	set 0, (iy++),e	set 0, (iy++),h	set 0, (iy++),l	set 0, (iy++)	set 0, (iy++),a	set 1, (iy++),b	set 1, (iy++),c	set 1, (iy++),d	set 1, (iy++),e	set 1, (iy++),h	set 1, (iy++),l	set 1, (iy++)	set 1, (iy++),a
D	set 2, (iy++),b	set 2, (iy++),c	set 2, (iy++),d	set 2, (iy++),e	set 2, (iy++),h	set 2, (iy++),l	set 2, (iy++)	set 2, (iy++),a	set 3, (iy++),b	set 3, (iy++),c	set 3, (iy++),d	set 3, (iy++),e	set 3, (iy++),h	set 3, (iy++),l	set 3, (iy++)	set 3, (iy++),a
E	set 4, (iy++),b	set 4, (iy++),c	set 4, (iy++),d	set 4, (iy++),e	set 4, (iy++),h	set 4, (iy++),l	set 4, (iy++)	set 4, (iy++),a	set 5, (iy++),b	set 5, (iy++),c	set 5, (iy++),d	set 5, (iy++),e	set 5, (iy++),h	set 5, (iy++),l	set 5, (iy++)	set 5, (iy++),a
F	set 6, (iy++),b	set 6, (iy++),c	set 6, (iy++),d	set 6, (iy++),e	set 6, (iy++),h	set 6, (iy++),l	set 6, (iy++)	set 6, (iy++),a	set 7, (iy++),b	set 7, (iy++),c	set 7, (iy++),d	set 7, (iy++),e	set 7, (iy++),h	set 7, (iy++),l	set 7, (iy++)	set 7, (iy++),a

C Durée des opcodes du Z80 sur CPC

La table suivante indique la durée des instructions du Z80, exprimée en nombre de NOPs équivalents, ce qui est spécifique au CPC. Par exemple, ADD A,(HL) a la même durée que 2 NOPs.

- r : 8 bits register (A,B,C,D,E,H,L)
- rr : 16 bits register (AF,BC,DE,HL,SP)
- d : 8 bits data (0..255)
- dd : 16 bits data
- b : bit (0..7)
- cond : Condition (CC,Z,M,NC,NZ,P,PO,PE)

Les instructions manipulant le registre IY sont équivalentes à celles qui utilisent IX, elles ne sont pas indiquées ici.

Opcode	Durée	Opcode	Durée	Opcode	Durée
ADC r,r	1	CP IXL	2	INC (IX+d)	6
ADC A,(HL)	2	CP (IX+d)	3	IND	5
ADC A,n	2	CPD	4	INI	5
ADC HL,rr	4	CPDR	6 / 4	INIR	6 / 5
ADC HL,SP	4	CPI	4	INDR	6 / 5
ADC A,IXH	2	CPIR	6 / 4	JP dd	3
ADC A,IXL	2	CPL	1	JP cond,dd	3
ADC A,(IX+d)	3	DAA	1	JP (HL)	1
ADD r,r	1	DEC r	1	JP (IX)	2
ADD A,(HL)	2	DEC rr	2	JR d	3
ADD A,d	2	DEC (HL)	3	JR C,d	3 / 2
ADD HL,dd	3	DEC IX	3	JR NC,d	3 / 2
ADD IX,rr	4	DEC IXH	2	JR NZ,d	3 / 2
ADD IX,IX	4	DEC IXL	2	JR Z,d	3 / 2
ADD IX,SP	4	DEC (IX+d)	6	LD r,r	1
ADD A,IXH	2	DI	1	LD r,d	2
ADD A,IXL	2	DJNZ d	4 / 3	LD A,(rr)	2
ADD A,(IX+d)	3	EI	1	LD r,(HL)	2
AND r	1	EX AF,AF'	1	LD (rr),A	2
AND d	2	EX DE,HL	1	LD SP,HL	2
AND (HL)	2	EX (SP),HL	6	LD r,IXH	2
AND IXH	2	EX (SP),IX	7	LD r,IXL	2
AND IXL	2	EXX	1	LD SP,IX	3
AND (IX+d)	3	HALT	1	LD rr,dd	3
BIT r	2	IM 0	2	LD (HL),d	3
BIT (HL)	3	IM 1	2	LD A,R	3
BIT b,(IX+d)	6	IM 2	2	LD R,A	3
BIT b,(IX+d),r	6	IN A,(d)	3	LD A,I	3
CALL dd	5	IN r,(C)	4	LD I,A	3
CALL cond,dd	5 / 3	INC r	1	LD IXH,d	3
CCF	1	INC rr	2	LD IXL,d	3
CP r	1	INC (HL)	3	LD A,(dd)	4
CP d	2	INC IX	3	LD (dd),A	4
CP (HL)	2	INC IXH	2	LD IX,dd	4
CP IXH	2	INC IXL	2	LD HL,(dd)	5

Opcode	Durée	Opcode	Durée	Opcode	Durée
LD BC, (dd)	6	RET	3	SCF	1
LD DE, (dd)	6	RET cond	4 / 2	SET b, r	2
LD (dd), HL	5	RETN	4	SET b, (HL)	4
LD r, (IX+d)	5	RETI	4	SET b, (IX+d)	7
LD (dd), rr	6	RL r	2	SET b, (IX+d), r	7
LD IX, (dd)	6	RL (HL)	4	SLA r	2
LD (dd), IX	6	RL (IX+d)	7	SLA (HL)	4
LD (IX+d), d	6	RL (IX+d), r	7	SLA (IX+d)	7
LD (dd), SP	6	RLC r	2	SLA (IX+d), r	7
LD SP, (dd)	6	RLC (HL)	4	SLL r	2
LDD	5	RLC (IX+d)	7	SLL (HL)	4
LDI	5	RLC (IX+d), r	7	SLL (IX+d)	7
LDDR	6 / 5	RLCA	1	SLL (IX+d), r	7
LDIR	6 / 5	RLA	1	SRA r	2
NEG	2	RLD	5	SRA (HL)	4
NOP	1	RR r	2	SRA (IX+d)	7
OR r	1	RR (HL)	4	SRA (IX+d), r	7
OR d	2	RR (IX+d)	7	SRL r	2
OR (HL)	2	RR (IX+d), r	7	SRL (HL)	4
OR IXH	2	RRA	1	SRL (IX+d)	7
OR IXL	2	RRC r	2	SRL (IX+d), r	7
OR (IX+d)	3	RRC (HL)	4	SUB r	1
OUT (d), A	3	RRC (IX+d)	7	SUB d	2
OUT (C), r	4	RRC (IX+d), r	7	SUB (HL)	2
OUT (C), 0	4	RRD	5	SUB IXH	2
OUTD	5	RRCA	1	SUB IXL	2
OUTI	5	RST d	4	SUB (IX+d)	3
OTDR	6 / 5	SBC A, r	1	XOR r	1
OTIR	6 / 5	SBC A, d	2	XOR d	2
POP rr	3	SBC A, IXH	2	XOR (HL)	2
POP IX	4	SBC A, IXL	2	XOR IXH	2
PUSH rr	4	SBC A, (HL)	2	XOR IXL	2
PUSH IX	5	SBC A, (IX+d)	3	XOR (IX+d)	3
RES b, r	2	SBC HL, rr	4		
RES b, (HL)	4	SBC HL, SP	4		
RES b, (IX+d)	7				
RES b, (IX+d), r	7				

D Licenses

RASM Ce logiciel (binaire et code source) ainsi que sa documentation utilisent la licence MIT "expat".

Copyright ©BERGÉ Édouard (roudoudou)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation/source files of RASM, to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. The Software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the Software.

LZ4 cruncher (sources were modified)

BSD 2-Clause License (<http://www.opensource.org/licenses/bsd-license.php>)
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You can contact the author at :

- LZ4 homepage : <http://www.lz4.org>
- LZ4 source repository : <https://github.com/lz4/lz4>

ZX7 cruncher (sources were modified)

(c) Copyright 2012 by Einar Saukas. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The name of its author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Exomizer cruncher (sources were modified)

Copyright (c) 2005 Magnus Lind.

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software, alter it and re-distribute it freely for any non-commercial, non-profit purpose subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any distribution.
4. The names of this software and/or its copyright holders may not be used to endorse or promote products derived from this software without specific prior written permission.

Index

SYMBOLS, 5

ACOS, 15
ALIGN, 10
AMSDOS, 28
AND, 15
AS80, 7
ASIN, 15
ASSERT, 17
ATAN(), 15
Audio, 25

BANK, 27, 29, 32, 33
BANKSET, 30, 32
BREAK, 18
BREAKPOINTS, 31
Breakpoints, 31
BRK, 21, 31
BUILDSNA, 29

Cartouches, 29
CASE, 18
CHARSET, 12
COS(), 15

DAMS, 21
DEFAULT, 18
DEFB, 10
DEFI, 11
DEFM, 10
DEFR, 11
DEFS, 11
DEFW, 11
DSK, 28

ELSE, 18
ENDIF, 18
ENDM, 20
ENDSTRUCT, 22
ENDSWITCH, 18
EQU, 14
Exomizer, 26

FAIL, 17

HI(), 10, 15

IF, 18
IFDEF, 18
IFNDEF, 18

IFNOT, 18
IFNUSED, 18
IFUSED, 18
INCBIN, 24
INCL48, 26
INCL49, 26
INCLEXO, 26
INCLUDE, 24
INCLZ4, 26
INCZX7, 26

Labels, 21
LET, 32
LIMIT, 10
LIST, 32
LO(), 15
LZ4, 25
LZ48, 25
LZ49, 25
LZCLOSE, 25
LZEX0, 25
LZX7, 25

MACRO, 20
MAXAM, 7
MEND, 20

OR, 15
ORG, 9

PAGE, 27
PAGESET, 27
PRINT, 17
PROTECT, 10

REND, 19
REPEAT, 19, 21
REPEAT_COUNTER, 19
RUN, 30

SAVE, 28
SETCPC, 30
SETCRIC, 31
SIN(), 15
SIZEOF, 22
Snapshots, 29, 31
STOP, 17
STR, 12
STRUCT, 22
SWITCH, 18

Symbols, 31

UNDEF, 18

UNTIL, 19, 21

UZ80, 7

Variables, 14

WAV, 25

WEND, 19

WHILE, 19, 21

WHILE_COUNTER, 19

WRITE DIRECT, 32

XOR, 15

ZX, 33