RASM

roudoudou Assembler
v0.75

Documentation RASM v0.75 (c) Édouard BERGÉ 2017

Sommaire

Introduction

Il y a 18 ans, j'avais programmé un assembleur/désassembleur appelé Zasm/Dizasm. En réalité c'était surtout une démonstration qu'il était possible de faire un assembleur mono-passe. J'ai un peu utilisé le désassembleur sur des projets personnels. Les sources ont été publié mais la diffusion a été si confidentielle que j'ai eu du mal à les récupérer à nouveau sur mon disque dur.

Depuis tout ce temps, un autre assembleur du nom de Zasm est sorti, créé par une équipe qui développe pour le ZX spectrum, un autre ordinateur équipé du Z80.

Vu que tout le code de cet assembleur est nouveau (à part quelques concepts), c'est l'occasion de le baptiser avec un nouveau nom: Rasm.

Installation

Rasm est un exécutable indépendant, il s'utilise tel que, sans installation aucune.

Compilation

compilation Linux:

```
cc rasm_v075.c -O2 -lm -lrt -march=native
mv a.out rasm
strip rasm
```

compilation Windows avec Visual Studio:

```
cl.exe rasm_v075.c -O2
```

compilation Morphos:

```
ppc-morphos-gcc-5 -O2 -c -o rasm rasm_v075.c
strip rasm
```

Comportement de Rasm

Rasm essaie d'être simple d'utilisation. En ce sens il va produire des noms de fichier par défaut, déterminer la quantité de mémoire à enregistrer ou même créer un fichier cartouche si les banques ROM ont été sélectionnées lors de l'assemblage.

Rasm permet l'utilisation de plusieurs ORG au sein d'un même espace mémoire. Par contre il n'autorise pas de ré-écrire sur les mêmes adresses mémoire. À chaque nouveau ORG, Rasm contrôle qu'aucune zone de code écrite ne se chevauche.

Si vous avez besoin de générer plusieurs morceaux de code à la même adresse, vous avez deux possibilités. Soit vous utilisez le paramètre <output> de la fonction ORG pour écrire ce code ailleurs, soit vous pouvez créer à tout moment un nouvel espace mémoire avec la fonction WRITE DIRECT -1,-1,#C0

La sauvegarde forcée d'espaces mémoire avec la fonction SAVE désactive l'écriture automatique de fichier ou cartouche. Il reste possible de forcer l'écriture cartouche avec la directive BUILDCPR.

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,MOD seront convertis en un seul caractère approchant la syntaxe du C.

Lorsqu'une directive de lecture ne fait pas référence à un chemin absolu, le répertoire racine au chemin relatif est celui du fichier en cours.

Utilisation de la ligne de commande

La syntaxe de base est: rasm.exe <fichier à assembler> [options]

Exemple: `rasm.exe monfichier.asm -s`

Options de fichier

-o <radix de fichier> définir un nom commun pour chaque type de fichier en sortie, quel que soit son type (.bin, .cpr, .sym). La valeur par défaut est “rasmoutput”.

`rasm.exe source.asm -o output -s` produira les noms suivants selon le type: `output.bin, output.cpr, output.sym`

-ob <nom de fichier binaire> définir le nom complet pour le fichier de sortie binaire.
-oc <nom de fichier cartouche>définir le nom complet pour le fichier de sortie cartouche.
-oi <nom de fichier snapshot> définir le nom complet pour le fichier de sortie snapshot.
-os <nom de fichier symbole> définir le nom complet pour le fichier de sortie des symboles.
-ok <nom de fichier breakpoint>définir le nom complet pour le fichier de sortie des points d'arrêt.
-no désactiver toute écriture de fichier

Options d'export sur EDSK

-eo écraser les fichiers sur l'image disque si il y a conflit de nom.

Options de symboles

-s exporter les symboles au format Rasm
-sp exporter les symboles au format Pasmo
-sw exporter les symboles au format Winape
-ss exporter les symboles dans un snapshot
-eb exporter les points d'arrêt

-sl option additionnelle aux trois précédentes qui permet d'exporter aussi les symboles locaux aux macros ou boucles de répétition.
-sv option additionnelle aux trois précédentes qui permet d'exporter aussi les variables.
-sq option additionnelle aux trois précédentes qui permet d'exporter aussi les alias EQU.
-sa option équivalente à -sl -sv -sq

- L'émulateur Arnold est compatible avec Rasm et Pasmo.
- L'émulateur Winape n'est pas capable de prendre en charge des labels trop longs!

-l <fichier label> importer un fichier de labels pour l'assemblage au format Rasm, Pasmo ou Winape

Options de compatibilité

-m mode de compatibilité avec maxam

- calculs réalisés en entiers 16 bits non signés avec un mauvais arrondi
- les comparaisons se font avec l'opérateur = simple.

-ass mode de compatibilité AS80

- calculs réalisés en entiers 32 bits avec un mauvais arrondi
- les déclarations DEFB,DEFW,DEFI 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

Options développeur

-v mode verbeux, affiche des informations et statistiques sur l'assemblage
-d produit des informations lors du pré-processing
-a produit des informations lors de l'assemblage
-n affiche les licences tierces

Format du code source

Généralités

L'assembleur n'est pas du COBOL, il est inutile d'indenter vos sources avec Rasm, autrement que pour faire joli. Il n'est pas nécessaire de séparer un label d'une instruction ou d'un autre label par le caractère deux points, bien qu'il soit possible de le faire. La conséquence directe de cette écriture libre implique une différence avec les assembleurs de conception obsolète. Il n'est pas possible (et heureusement) de créer un label qui ait le même nom qu'une directive ou instruction Z80.

Le défaut de cette liberté est que l'assembleur 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.

Les fichiers peuvent être lus au format windows ou unix, une conversion interne et transparente sera réalisée en interne.

Rasm n'est pas sensible à la casse, toutes les lettres sont converties en majuscules en interne. Ne soyez pas surpris de ne voir que des majuscules dans les messages d'erreur.

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 n'y a pas de commentaire multi-lignes

Valeurs litérales

Rasm interprète les valeurs numériques suivantes:

- 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 quote.
- En valeur “interne” à une variable ou un label, si la litérale 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 ou DEFL, 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*.

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.

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

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, circonflex, le pourcent, le dièse, le paragraphe, les chevrons et les deux types de quotes.

Fichiers

RASM utilise en interne le modèle de fichiers Unix qu'il converti automatiquement au besoin. En utilisant uniquement des chemins relatifs (et des noms de fichiers que Windows peut comprendre), on peut tout à fait avoir un code qui compile à la fois sous Unix et Windows sans faire de modification.

La règle de gestion des chemins relatifs est simple. Tout chemin relatif a pour origine le répertoire du fichier dans lequel il a été lu.

Spécificités sur les labels

À l'intérieur d'une boucle (REPEAT/WHILE/UNTIL) ou dans 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 '@'.

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 déclaration désuète d'un label en le préfixant d'un point. L'appel à ce label se fera sans le point du début.

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.

Tag spécifiques aux labels

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

Le préfixe {PAGE} devant un label (exemple: {PAGE}monlabel) permet de récupérer la valeur 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 → #C4

Le préfixe {PAGESET} devant un label (exemple: {PAGESET}monlabel) permet de récupérer la valeur 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 → #C2

Expressions

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

Opérateurs de calcul

- * multiplication
- / division
- + addition
- - soustraction
- & opérateur logique ET
- | opérateur logique OU
- ^ opérateur logique OU exclusif
- && opérateur booléen ET
- || opérateur booléen OU
- % ou § modulo (utilisez MOD si votre clavier ne dispose pas de ce caractère)
- << multiplication par la puissance n de deux
- >> division par la puissance n de deux
- AND, OR, XOR, MOD en mode maxam
- 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
- floor() conversion au nombre entier directement inférieur
- abs() valeur absolue
- ln() logarithme népérien
- log10() logarithme base 10
- exp() exponentielle
- sqrt() racine carrée

Opérateurs de comparaison

- == égalité (ou un seul = en mode maxam)
- != différent de
- <= inférieur ou égal
- >= supérieur ou égal
- < inférieur
- > supérieur

Spécificités sur les expressions

~~Les opérateurs de comparaison ne doivent pas être imbriqués dans des parenthèses:~~

~~syntaxe correcte:~~ `IF (4+mavar)*5==monresultat`

~~syntaxe incorrecte:~~ `IF ((4+mavar)*5==monresultat)`

Variables statiques ou alias

Il est possible de créer des alias avec la directive EQU. Ces alias ne peuvent pas être modifiés une fois qu'ils sont définis.

Variables dynamiques

Rasm autorise un nombre illimité de variables pour des calculs internes.

Syntaxe: `mavariable=5` ou `LET mavariable=5`

Ces variables peuvent être utilisées comme compteur de boucle ou comme offset lors d'une boucle de répétition.

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
```

Directives

Une directive n'est pas une fonction, c'est un mot clef qui doit être séparé de ses paramètres par au moins un espace.

Syntaxe correcte: `ASSERT (4*mavar)`
Syntaxe incorrecte: `ASSERT(4*mavar)`

ASSERT <condition>

Vérifier une condition et arrêter l'assemblage si la condition est fausse.

EQU

Créer un alias

Exemples:
```
mavariable EQU 5
monautre EQU mavariable*2
```

ALIGN <valeur>

Aligner le code produit sur une valeur multiple du paramètre valeur.

Exemples:
`ALIGN 2` pour aligner sur une adresse paire
`ALIGN 256` pour aligner sur le poids fort d'adresse

AMSDOS

Ajoute un entête Amsdos au fichier binaire produit automatiquement par Rasm. Note: Cet entête n'est pas ajouté lors d'un SAVE.

BREAKPOINT <adresse>

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. Les points d'arrêt peuvent être exportés sous forme de fichier brut ou dans les snapshots (compatible avec les émulateurs ACE et Winape).

Tout label qui commence par le préfixe BRK ou @BRK génère à la fois un label et un point d'arrêt.

Il est possible de donner en paramètre une adresse afin de mettre le point d'arrêt n'importe où en mémoire.

BUILDCPR

Pour forcer l'écriture de la cartouche lorsqu'on a utilisé une instruction SAVE.

BUILDSNA <V2>

Pour forcer l'écriture d'un snapshot (v3). 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.

Le snapshot généré par Rasm inclu un écran de taille classique (le même que sous Basic), toutes les encres en bleu foncé sauf l'encre 1 en jaune vif. Les 3 voies audio sont désactivées, les roms désactivées et le mode d'interruption 1.

Exemple: créer un snapshot avec différents types de sélection mémoire

```
buildsna ; il est conseillé de commencer par déclarer le SNA pour
; éviter d'avoir un problème de limite à 512K

bankset 0 ; assembler dans les premiers 64K de la machine
org #1000
run #1000 ; le snapshot s'exécutera en #1000

ld b,#7F
ld a,{page}mydata ; récupère l'ordre gate array de pagination mémoire
out (c),c
ld a,(mydata)
jr $

bank 6 ; choisir la troisième bank du deuxième jeu de 64K
nop
mydata defb #DD

bank ; sans paramètre, créer un espace mémoire indépendant qui ne
; sera pas enregistré dans le snapshot

pouet
repeat 10
cpi
rend
camion

save"autrechose",pouet,camion-pouet
```

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 comme suit:

```
buildsna v2
```

BANK [<numéro de page ROM, numéro de page RAM>]

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.

BANKSET <numéro de bloc de 64K>

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 fonction active automatiquement la génération de snapshot.

IF, IFNOT, ELSE, ELSEIF, ENDIF

Permet d'écrire du code conditionnel.

Exemple:

```
CODE_PRODUCTION=1
[...]
IF CODE_PRODUCTION
OR #80
ELSE
PRINT 'Version de test'
ENDIF
```

IFDEF, IFNDEF <variable>

Ces deux directives conditionnelles permettent de tester l'existence d'une variable.

LZ48 / LZ49 / LZ4 / LZX7 / LZEXO

Ouvrir un segment de code compressé en LZ48,LZ49, LZ4, ZX7 ou Exomizer. Le code produit sera compressé après assemblage et le code suivant le segment compressé sera relogé.

Il n'est pas possible d'appeler un label situé après un segment compressé depuis le code compressé pour des raisons évidentes dûes aux aléas de la compression. Une erreur s'affichera expliquant pourquoi.

Limitations:

- Pour le moment, le code avant compression et les éventuelles données situées après ne peuvent pas dépasser l'espace d'adressage de 64Ko.
- Il n'est pas possible d'imbriquer les segments compressés.

LZCLOSE

Fermer un segment compressé.

READ / INCLUDE 'fichier à lire'

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. Attention à ce que vous faites.

INCBIN 'fichier à lire'[,offset[,size[,offset étendu[,OFF]]]]

Lire un fichier binaire. Les données lues seront directement injectées. 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.

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 de zéro 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 ; lecture en #4000
org #4001
defb #BB ; écrasement du deuxième octet sans erreur
```

INCL48 / INCL49 / INCLZ4 / INCZX7 / INCEXO 'fichier à lire'

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

LIMIT <adresse limite>

Imposer une limite plus basse à l'écriture de l'assemblage. Par défaut, la limite est de 65536 mais on peut avoir besoin de ne pas dépasser une certaine valeur. Pour protéger une zone définie, il vaut mieux utiliser la fonction PROTECT.

ORG <adresse logique>[,<adresse d'écriture du code>]

Assembler le code à une adresse spécifique. On peut optionnellement choisir d'assembler le code à une adresse, mais l'écrire à un autre endroit avec le deuxième paramètre.

PROTECT <adresse début>,<adresse fin>

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

STR '<chaine>','<chaine2>',...

Similaire à DEFB '<chaine>', le dernier caractère de chaque chaine a son bit 7 forcé à 1 (OR #80 sur le dernier octet de chaque chaine).

STOP

Arrêter l'assemblage.

PRINT '<string>',variables,expression,...

Écrire du texte, variables ou expressions lors de l'assemblage. Il est possible de formater les variables en préfixant la variable 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 variabke 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.

Sans préfixe la variable ou expression est affichée en nombre flottant.

RUN <valeur>

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.

LIST / NOLIST / LET

Fonctions ignorées, pour compatibilité avec Winape.

WHILE / WEND

Répète un bloc d'instructions tant que la condition est vérifiée. Il est possible de consulter à tout moment la variable while_counter pour déclencher des évènements en fonction du numéro d'itération.

```
Exemple:

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

La boucle sera exécutée 10 fois de suite produisant la sortie texte suivante:

```
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)
```

REPEAT <n> / 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. Il est possible à tout moment de consulter le numéro d'itération du repeat courant avec la variable repeat_counter.

Exemples:

```
repeat 10
ldi
print repeat_counter
rend

cpt=10
repeat
ldi
cpt=cpt-1
until cpt>0
```

WRITE DIRECT <rom basse>,<rom haute>,<RAM>

En spécifiant une rom basse (entre 0 et 7) ou une rom haute (entre 0 et 31), cette instruction remplace la directive BANK et a le même effet.

En spécifiant uniquement l'adresse RAM (n'importe quelle valeur) et en désactivant les numéros de rom avec la valeur -1, on créé à chaque appel un nouvel espace mémoire. On peut ainsi assembler plusieurs code au même emplacement mémoire, mais dans un espace différent.

Exemple:

```
;par defaut un espace memoire est cree, ORG en zero
defs 65536,0
WRITE DIRECT -1,-1,#C0
defs 65536,0
; cree un nouvel espace memoire peu importe
; la valeur de la BANK, ici #C0 deux fois
WRITE DIRECT -1,-1,#C0
; ecriture au meme endroit ne genere pas d'erreur
defs 65536,0
```

Rasm aura créé trois espaces mémoire. L'espace par défaut et un espace supplémentaire à chaque WRITE DIRECT. Il n'y a pas de limite au nombre d'espace mémoire dynamique autre que la mémoire totale de votre système.

Il n'est pas possible de revenir sur un espace mémoire dynamique une fois qu'on l'a quitté.

SAVE 'fichier binaire à écrire',<adresse>,<taille>,<type>,...

Enregistre le fichier binaire correspondant à la mémoire adresse jusqu'à adresse+taille. 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.

En conséquence de quoi il n'est pas possible d'enregistrer des états intermédiaires d'assemblage.

Enregistrer un fichier binaire

```
SAVE 'monfichier.bin',debut,taille
```

Enregistrer un fichier binaire avec entête AMSDOS

```
SAVE 'monfichier.bin',debut,taille,AMSDOS
```

Enregistrer un fichier binaire sur une image de disquette

```
SAVE 'monfic.bin',debut,taille,DSK,'fichierdsk.dsk'
```

Le nom de fichier sera automatiquement tronqué et mis en majuscules si il n'est pas conforme aux limitations de l'AMSDOS.

SETCPC <modèle>

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

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

SETCRTC <modèle>

Choisir le modèle de CRTC quand on enregistre un snapshot. Les valeurs autorisées vont de 0 à 4.

CHARSET 'chaine',<valeur> | <code>,<valeur> | <début>,<fin>,<valeur>

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

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

Cette fonction est compatible Winape.

SWITCH, CASE, BREAK, DEFAULT, ENDSWITCH

La syntaxe est une mimic du case en C, avec la particularité de pouvoir écrire plusieurs CASE qui ont la même valeur, ce qui donne plus de souplesse au code conditionnel.

Dans cet exemple, l'appel de la macro avec 5 assemblera toutes les chaines de defb avec 'oui' ou 'encore oui':

```
macro ouioui mavar
switch {mavar}
    nop ; pas assemblé car hors case
    case 3
        defb 'non'
    case 5
        defb 'oui'
    case 7
        defb 'encore oui'
        break
    case 8
        defb 'non'
    case 5
        defb 'encore oui'
        break
    default
        defb 'non'
endswitch
mend
```

Macros

Rasm supporte les macros avec chevrons (compatible avec Winape). Il est possible de faire de l'assemblage conditionnel avec les macros car à chaque appel de macro, le code d'origine est inséré, les paramètres substitués et enfin le code est interprété de façon classique.

Exemple pour une écriture longue distance générique (sauf pour B ou C):

```
macro LDIXREG registre,dep
if {dep}<-128 || {dep}>127
    push bc
    ld bc,{dep}
    add ix,bc
    ld (ix+0),{registre}
    pop bc
else
    ld (ix+{dep}),{registre}
endif
mend
```

Note 1: Le marqueur de fin de macro peut être indifférement MEND ou ENDM

macros sans paramètre

En cas de faute de frappe, pour éviter que la macro mal orthographiée soit remplacée par un label, il est conseillé d'utiliser le paramètre (VOID) à chaque appel. Ainsi, si la macro est mal orthographiée, la présence du (VOID) déclenchera une erreur.

```
macro sansparam
nop
mend

sansparam (void) ; appel sécurisé de la macro
```

Appel de macro en paramètre statique ou dynamique

Les macros de Rasm sont (en logique interne) du code qui est remplacé au vol lors de l'assemblage. Il y a deux possibilités de remplacement pour les paramètres. Soit le paramètre est copié tel que, soit il peut être calculé et c'est sa valeur calculée qui sera injecté dans la macro.

```
macro test myarg
defb {myarg}
mend

repeat 2
test repeat_counter
rend
repeat 2
test {eval}repeat_counter
rend
```

Dans la première boucle, c'est la ligne `defb repeat_counter` qui sera développée tandis que le second appel, avec le paramètre précédé du tag `{eval}`, c'est la première valeur de repeat_counter qui sera développée, entrainant deux `defb` identiques.

Instructions

Toutes les instructions documentées et non documentées sont supportées.

L'adressage 8 bits des registres d'indexe IX et IY se fait indifféremment avec lx ou xl, hx ou xh, etc.

Les instructions complexes s'écrivent de la façon suivante:

```
res 0,(ix+0),a
bit 0,(ix+0),a
sll 0,(ix+0),a
rl  0,(ix+0),a
rr  0,(ix+0),a
```

Syntaxe des instructions de port non documentées:

```
out (<n>),a ; <n> est une valeur 8 bits
in a,(<n>)
in 0,(c) ou in f,(c)
```

Syntaxe spéciales autorisées:

```
push bc,de,hl ; → push bc : push de : push hl
nop 4         ; → nop : nop : nop : nop
```

Limitations

- Il n'est pas possible d'utiliser la mnémonique d'une instruction Z80 comme un label.
- Il n'est pas possible d'utiliser le même nom pour un label, une variable ou un alias.
- Rasm n'est pas sensible à la casse.
- Les blocs de code à compresser dynamiquement ne peuvent excéder 64K.

Exemples de programmation avec RASM

Récupérer le numéro de bank d'un label pour connecter une ROM

L'usage du préfixe {BANK} est pratique quand on prévoit d'appeler des routines susceptibles d'être relogées dans n'importe quelle banque.

```
Bank 0
ld a,{bank}maroutine ; sera assemblé LD A,1
call connect_bank    ; il faudra faire la votre!
jp maroutine

bank 1
defb 'coucou'
maroutine
jr $
```

Création d'un snapshot

```
Buildsna
bankset 0 ; assembler dans l'ensemble des 64k
org #1000
; programme
nop

bank 4 ; assembler dans la première page des 64K supérieurs d'un 6128
org #4000
defb 1
; comme on connectera cette bank en #4000 il est plus simple de la commencer
; à cette adresse mais elle ne fera bien que 16K car c'est une BANK et pas un BANKSET
```

Sauvegarde d'un programme sur un DSK

Rasm est capable de créer ou de modifier des images disquette au format DSK ou EDSK

```
org #8000

debutcode

ld hl,hello
printcharloop
ld a,(hl)
or a
ret z
call #BB5A
inc hl
jr printcharloop
hello defb 'Hello world',13,10,0

fincode

save"hello.bin",debutcode,fincode-debutcode,DSK,"hello.dsk"
```