Assembleur Mini-HowTo

par François-René Rideau rideau@ens.fr Version 0.3c, 15 Juin 1996

(Adaptation française par Eric Dumas dumas@freenix.fr, 28 Octobre 1996).

1 Préliminaires

Ce mini-howto est destiné à présenter les possibilités de développer en assembleur avec des processeurs x86 en utilisant *uniquement* des outils librement diffusés.

2 Copyright

3 Note importante

Il s'agit toujours d'une *première version incomplète* de ce document. Vous (oui, c'est à vous que je parle en ce moment, donc écoutez un peu !) êtes cordialement invités à :

- me poser des questions ;
- y répondre ;
- corriger les réponses données ;
- donner des pointeurs sur d'autres logiciels ;
- insulter le mainteneur actuel de ce document ;
- **de prendre ma place concernant la maintenance de la FAQ** car j'ai pas mal d'autres choses à faire.

Pour n'importe quel de ces points, n'hésitez-pas à me contacter à l'adresse électronique `rideau@ens.fr`.

Peut-être arriverons-nous à convaincre *Raymond Moon* d'ajouter une section à sa FAQ du forum Usenet `comp.lang.asm.x86`...?

4 Introduction

Ce document a pour but de répondre aux questions fréquemment posées par des gens qui développent ou souhaitent développer en assembler 32 bits pour processus x86, en utilisant des assembleurs libres, et en particulier sous le système d'exploitation Linux. Ce document peut également faire référence à d'autres documents présentant des assembleurs qui ne soient pas libres, non 32 bits ou tout simplement non destinés aux processeurs x86 car ce n'est pas sa vocation.

Comme l'intérêt principal des langages assembleur est d'écrire et de construire les fondations des systèmes d'exploitation, des langages et des jeux là où les compilateurs C échouent car incapables de fournir la même souplesse (la performance est rarement un problème), les développeurs stressent lors de tels développements.

4.1 Comment utiliser ce document

Ce document contient les réponses aux questions qui sont les plus souvent posées. Vous trouverez fréquemment des liens (URL) sur des logiciels ou des documentations. Observez s'il vous plaît que généralement les endroits où sont situés ces produits sont recopiés sur d'autres sites, qui peut être bien plus proche de chez vous. En accédant à ces sites plus proches, vous évitez que du trafic inutile sur Internet se produise, tout en vous économisant du temps précieux. En particulier, il existe de gros sites aux quatre coins de la planète qui recopient d'autres répertoires de données fréquemment utilisés. Vous devriez apprendre et noter quels sont ces sites qui sont proches de chez vous. Parfois, la liste des miroirs est donnée dans un fichier ou dans le message de connexion. Suivez ce petit conseil. Sinon, vous pouvez toujours demander à `archie` où se trouvent les programmes que vous recherchez...

La version la plus récente de ce document peut être consultée à l'URL suivante : *http://www.eleves.ens.fr:8080/home/rideau/Assembly* mais ce qui se trouve dans l'arborescence des HowTo Linux devrait normalement être à jour également : *ftp://sunsite.unc.edu/pub/linux/docs/HOWTO/*. Enfin, en principe ! (N.d.T : les versions françaises se trouvent sur le site : *ftp://ftp.ibp.fr/pub/linux/french/HOWTO*, et à l'URL *http://www.freenix.fr/linux*)

4.2 Autres documents de références

- Si vous ne savez pas ce que sont les logiciels *libres*, vous devriez lire attentivement la GPL (*GNU General Public License*), qui est utilisée dans bon nombre de logiciels libres, et en tant que modèle pour la plupart. Elle est généralement diffusée dans un fichier appelé `COPYING`. Une version pour les bibliothèques se trouve dans le fichier `COPYING.LIB`. La documentation fournie par la FSF (*Free Software Foundation*) peut également vous aider.
- Plus particulièrement, l'aspect intéressant des logiciels libres, c'est qu'ils sont livrés avec leurs sources que vous pouvez lire et éventuellement corriger. Vous pouvez même les adapter. Lisez votre licence attentivement et conformez-vous en.
- Il existe une FAQ pour le forum Usenet `comp.lang.asm.x86` qui donne certaines réponses générales concernant la programmation en assembleur avec les processeurs x86, et répond à certaines questions sur certains assembleurs Dos 16 bits commerciaux. Certaines de ces réponses peuvent également concerner les assembleurs libres 32 bits donc vous pourriez également avoir à la consulter... *http://www2.dgsys.com/˜raymoon/faq/asmfaq.zip*.

– Des FAQ ainsi que des documentations existent concernant votre plate-forme de développement favorite, quelle qu'elle soit. Vous pouvez vous y référer pour y consulter pour tout ce qui concerne les spécifiés de la plate-forme, et qui ne concerne pas directement la programmation en assembleur.

5 Assembleurs

5.1 Assembleur Gcc en ligne

Le très célèbre compilateur C/C++ Gnu (`gcc`) est un compilateur 32 bits très optimisé qui est au coeur du projet GNU, qui gère assez bien l'architecture x86, et qui inclue la possibilité d'insérer du code assembleur dans les programmes C de telle manière que l'allocation de registres peut être soit spécifiée, soit effectuée par `gcc`. `gcc` fonctionne sur la plupart des plate-formes existantes comme sous Linux, *BSD, VSTa, OS/2, *Dos, Win*, etc.

5.1.1 Où trouver Gcc ?

Le site principal de `gcc` est `ftp://prep.ai.mit.edu/pub/gnu/` ainsi que les autres applications du projet GNU. Toutefois, il existe bon nombre de miroirs (NdT : comme `ftp.ibp.fr/pub/gnu` en France).

Toutefois, les sources adaptés pour votre système d'exploitation favori ainsi qu'une version binaire toute prête peut être trouvée sur les sites ftp usuels.

Pour la version Linux de `gcc`, il suffit de regarder sur le site `http://www.linux.org.uk/`

Le portage le plus fameux de `gcc` est appelé *djgpp* et peut être trouvé dans l'arborescence de la plupart des sites ftp, et en particulier sur le site `http://www.delorie.com/djgpp/`.

Il existe également un portage de gcc sous OS/2 appelé *EMX* qui fonctionne également sous Dos. Consultez les sites *http://www.leo.org/pub/comp/os/os2/gnu/emx+gcc/* et *http://warp.eecs.berkeley.edu/os2/software/shareware/emx.html*.

5.1.2 Où trouver de la documentation concernant l'assembleur avec gcc ?

La documentation de GCC inclue des fichiers au format *texinfo*, que vous pouvez convertir au format *tex*, compiler avec le programme du même nom, et imprimer. Sinon, vous pouvez les consulter d'une manière interactive en utilisant les fichiers `info` d'*emacs*. Les fichiers `info` sont généralement trouvés dans toute bonne installation de gcc.

La partie à consulter est :

```
        C Extensions::Extended Asm::
```

```
Section
        Invoking GCC::Submodel Options::i386 Options::
```

Cette partie vous donne les contraintes concernant les noms de registres : `abcdSDB` correspondent respectivement au registres (aucune lettre pour %esp) :

```
%eax, %ebx, %ecx, %edx, %esi, %edi, %ebp
```

Une URL concernant ce document et cette section au format HTML se trouve sur : *http://www.cygnus.com/doc/usegcc_89.html*.

Les ressources concernant le *DJGPP Games* (ne concerne pas seulement les gens qui fouinent le code des jeux...) possède une page particulière sur l'assembleur : *http://www.rt66.com/˜brennan/djgpp/djgpp_asm.html*.

Enfin, il existe une page Web appelée *DJGPP Quick ASM Programming Guide* qui contient certaines URL, FAS, la syntaxe assembleur d'AT&T, certaines information concernant l'assembleur `inline` et la conversion de fichier objets en bibliothèques : *http://remus.rutgers.edu/˜avly/djasm.html*.

En fait, gcc dépend de `gas` du côté de l'assembleur et suit sa syntaxe (voir plus vas). N'oubliez pas que l'assembleur en ligne implique la mise entre quotes des signes pourcent pour qu'ils soient passés à `gas`. Lire la section traitant `gas`.

Vous trouverez beaucoup d'exemples très instructifs dans le sous-répertoire `linux/include/asm-i386/` des sources du système d'exploitation libre Linux.

6 Comment appeler gcc pour qu'il traite correctement mon code assembleur ?

Assurez-vous d'appeler `gcc` avec l'option « -O » pour activer les optimisations ainsi que l'assembleur en ligne. Si vous ne l'utilisez pas, votre code peut compiler, mais ne s'exécutera pas convenablement !

Plus généralement, les bonnes options de compilations sur des plate-formes x86 avec gcc sont :

```
gcc -O2 -fomit-frame-pointer -m386
```

– `-O2` est le bon niveau d'optimisation à utiliser. Le fait d'utiliser un niveau d'optimisation supérieur produit un code qui est beaucoup plus important, mais seulement un peu plus rapide : de telles optimisations massives peut être utiles uniquement pour de courtes boucles (s'il y en a), que vous pourriez éventuellement faire en assembleur de toute manière. Si vous avez besoin de telles optimisations, faîtes les uniquement pour les quelques routines qui en ont besoin.

– `-fomit-frame-pointer` autorise le code généré à ne pas gérer la maintenance particulièrement stupide des *frame pointers*, ce qui rend ainsi le code plus rapide et plus petit. Cette option permet également de libérer un registre pour des optimisations supplémentaires. Cette option toutefois réduit la facilité d'utilisation des outils de débogage (comme `gdb`), mais lorsque vous les utilisez, la vitesse et la taille du programme sont peu importants.

– `-m386` génère un code plus compacte, sans que l'on puisse mesurer une réelle perte de performance (vous pouvez remarquer que plus le code est petit, moins il y a d'opération d'entrées sorties sur disque et que donc l'exécution est plus rapide).

Pour optimiser encore plus la génération du code, l'option de compilation `-mregparm=2` et/ou la fonction correspondante peuvent être utiles mais peut poser pas mal de problèmes lors de l'édition de liens avec du code externe.

Notez que vous pouvez ajouter ces options en les rendant utilisables par défaut en modifiant le fichier `/usr/lib/gcc-lib/i486-linux/2.7.2/specs` (ou quelque soit sa place sur votre système).

6.1 gas

`gas` est l'assembleur GNU, qui est utilisé par `gcc`.

6.1.1 Où le trouver ?

Vous le trouverez au même endroit que `gcc`, dans un paquetage appelé `binutils`.

6.1.2 Qu'est-ce que la syntaxe AT&T

Etant donné que `gas` a été inventé pour être un compilateur unix 32 bits, il utilise la syntaxe standard AT&T, qui ressemble beaucoup à la syntaxe des assembleurs 680x0. Cette syntaxe n'est pas pire, ni meilleur que la syntaxe Intel. Elle est tout simplement différente. Lorsque vous aurez l'habitude de l'employer, vous la trouverez beaucoup plus régulière que la syntaxe Intel, bien qu'un peu ennuyeuse.

Un programme existe pour vous aider à convertir des programmes écrit avec la syntaxe *TASM* en une syntaxe *AT&T*. Récupérez le fichier *ftp://x2ftp.oulu.fi/pub/msdos/programming/convert/ta2asv08.zip*.

Un fichier `gas.doc` ou `as.doc` (pas très loin de l'endroit où vous trouverez `gas`, si ce n'est dans son paquetage lui-même) décrit la syntaxe.

Vous pouvez le trouver sur les sites ftp suivants :

```
ftp://sunsite.unc.edu/pub/linux/GCC/
ftp://sunsite.doc.ic.ac.uk/packages/linux/sunsite.unc-mirror/GCC/ (?)
ftp://ftp.ibp.fr/pub/gnu
```

A nouveau, les sources du système d'exploitation Linux sont de bons exemples. Regardez dans le répertoire `linux/arch/i386` les fichiers suivants :

- `kernel/entry.S`
- `kernel/head.S`
- `boot/compressed/head.S`
- `mathemu/*.S`

6.1.3 Mode limité 16 bits

`gas` est un assembleur 32 bits, conçu pour gérer les instructions 32 bits. Il ne gère que partiellement le mode 16 bits, qui consiste à transformer les préfixes 32 bits en instructions, de telle manière que vous écriviez du code 32 bits qui s'éxécute en mode 16 bits sur un processeur 32 bits. Dans les deux modes, il gère l'utilisation des registres 16 bits mais l'adressage 16 bits n'est pas supporté. Utilisez les directives `code16` et `code32` pour basculer d'un mode à l'autre. Remarquez que l'instruction assembleur en ligne

```
asm("code16\n")
```

autorisera `gcc` (32 bits) à générer du code qui s'exécutera en mode réel ! Vous êtes libres d'ajouter une gestion complète 16 bits si vous en avez besoin.

6.2 GASP

`gasp` est le préprocesseur `gas`. Il permet l'utilisation des macro-commandes ainsi que certaines syntaxes sympathiques de `gas`.

6.2.1 Où trouver gasp ?

Je ne sais pas vraiment. Regardez dans les répertoires du projet GNU (`prep.ai.mit.edu` et miroirs). Peut-être avec `gas` dans le paquetage `binutils` ?

6.2.2 Comme ça marche ?

Aucune idée ! Mais c'est livré avec sa propre documentation en texinfo donc vous avez juste à l'imprimer ou à les consulter... En ce qui me concerne, ça ressemble à un macro-assembleur.

6.3 AS86

`as86` est un assembleur pour 80x86, à la fois 16 et 32 bits, qui fait partie du Bruce Evans' C Compiler (BCC). Il utilise en grande partie la syntaxe Intel, bien qu'exprimant les modes d'adressage d'une manière différente.

6.3.1 Où récupérer as86 ?

Une version complètement dépassée de `as86` est distribuée par HJLu pour arriver à compiler le noyau Linux, dans un paquetage appelé `bin86` (à l'heure actuelle, version 0.3), disponible dans n'importe quel répertoire GCC pour Linux. Mais je ne conseille à personne de l'utiliser pour autre chose que pour compiler Linux. Cette version ne supporte qu'un format de fichier minix modifié, et possède certaines erreurs avec le mode 32 bits. Donc, ne vous en servez que pour compiler Linux.

Les versions les plus récentes sont diffusées avec la distribution FreeBSD. Je les récupère dans *ftp://ftp.ibp.fr/pub/FreeBSD/packages-2.1/development/bcc-95.3.12.tgz*. Mais le chemin peut être fonction de votre miroir et la version peut avoir changée. En autre chose, il gère le format Linux a.out et vous pouvez donc effectuer une édition de liens avec d'autres programmes existant, et/ou utiliser les outils standards du paquetage `binutil` pour manipuler vos données. Cette version peut co-exister sans aucun problème avec la version précédente.

La version de `bcc` du 12 mars 1995 ainsi que les version antérieur ont un petit problème qui rend tous les segments empilés ou dépilés 16 bits, ce qui est assez ennuyant lorsque l'on développe en mode 32 bits. Un patch est diffusé dans le projet *Tunes* : *http://www.eleves.ens.fr:8080/home/rideau/Tunes/* puis *files/tunes.0.0.0.25.src.tgz* ou le sous-répertoire `LLL/i386/` sous forme décompressée. Ce patch est également disponible via *http://www.eleves.ens.fr:8080/home/rideau/files/as86.bcc.patch.gz*. Bruce Evans a accepté ce patch donc s'il existe une version plus récente de `bcc` quelque part, ce patch devrait être inclu...

Note pour les utilisateurs de Dos (?) :

- pour le compiler sous Dos, il est possible que vous deviez définir `POSIX_HEADERS_MISSING` ;
- pour `bcc/as`, si vous n'utilisez pas `djgpp`, vous devrez renomer la variable « far » dans la fonction `mcall()` du fichier `mops.c`, car certains compilateurs Dos pensent que « far » est un mot réservé. Notez que vous devez réaliser l'édition de lien avec `typeconf.obj` du répertoire `bcc/ld`...
- pour `bcc/ld`, vous devrez avoir une copie des fichiers d'en-tête `a.out.h` et `ar.h`. `djgpp` les possède mais d'autres compilateurs C peuvent demander à ce que vous les récupériez (sous Dos, Linux, VSTa, etc).
- pour `bcc/ld`, vous devez définir la macro `BSD_A_OUT` pour tous les fichiers, et éditer le fichier `writebin.c` pour qu'il définisse `STANDARD_GNU_A_OUT` et ainsi inclure une copie du fichier `a.out.h` Linux avec un nom Dos utilisanle ;
- le fichier `a.out.h` Linux inclue à son tour le fichier `asm/a.out.h`, donc vous devez vous arranger à ce qu'il soit inclus également. Les compilateurs 16-bit demandent à ce que vous éditiez le fichier `asm/a.out.h` pour modifier un champ sur 24 bits en un ensemble de champs de moins de 24 bits (`ld` ne sait pas se servir de ce champ).
- je n'ai pas essayé `cc1`, mais il devrait être capable de le compiler également si vous voulez l'utiliser. Toutefois, vous devrez re-écrire une partie de `bcc` (le *frontend*), ou bien utiliser `cc1` directement car il se base sur le triptique `fork(),exec()` et `wait()` pour lancer `cc1`, `as`, et `ld` lors de la compilation.
- des version Dos précompilées peuvent être trouvées sur *http://www.eleves.ens.fr:8080/home/rideau/files/asld86.zip*. Si vous voulez les recompiler avec des compilateurs libres et me les envoyer, j'aprécierai beaucoup.

6.3.2 Où trouver de la documentation?

Les documentations sont inclues dans le paquetage `bcc`. Les pages de manuels sont également disponibles quelque part sur les sites FreeBSD. Dans le doute, les sources sont bien souvent de très bonnes sources de documentation : ce n'est pas très bien commenté mais le code est propre. Vous devriez jeter un coup d'oeil pour voir comment `as86` est utilisé dans `Tunes 0.0.0.25`...

6.3.3 Comment appeler l'assembleur ?

Voici l'entrée dans le Makefile GNU pour permettre à `bcc` de transformer les fichiers assembleurs `.s` en objets `a.out` et `.o` ainsi qu'en listing `.l` :

```
%.o %.l:        %.s
        bcc -3 -G -c -A-d -A-l -A$*.l -o $*.o $<
```

Supprimez les `%.l`, `-A-l` et `-A$*.l` si vous ne voulez pas les listings. Si vous voulez autre chose que du format `a.out`, alors consultez la documentation de `bcc` pour connaître les autres formats gérés ou utilisez le programme `objcopy` du paquetage `binutils`.

6.3.4 Que faire si je n'arrive plus à recompiler Linux avec cette nouvelle version ?

Linus étant particulièrement surchargé par le courrier, mon patch pour compiler Linux avec un as86 a.out n'a pas du lui parvenir... Maintenant, cela ne devrait pas être vraiment important : conservez votre as86 du paquetage bin86 dans le répertoire `/usr/bin`, et installez le bon programme `as86` comme étant `/usr/local/libexec/i386/bcc/as`. Vous n'aurez en fait jamais a appeler directement ce programme car `bcc` fait tout seul ces opérations, dont la conversion en a.out Linux lorsqu'il est appelé avec les bonnes options. Donc, on peut assembler des fichiers uniquement en utilisant `bcc` comme un *frontend* et pas en utilisant directement `as86`.

6.4 Les autres assembleurs

Ce sont d'autres possibilités dans le cas où les précédentes ne vous conviennent pas (pourquoi donc ?) que je ne vous recommande pas dans un cas normal mais qui peut s'avérer utile si l'assembleur fait partie de ce que vous ètes en train de concevoir (par exemple un système d'exploitation ou un environement de développement).

6.4.1 Assembleur Win32Forth

Win32Forth est une solution 32 bits *libre* FORTH qui tourne sans problème sous Win32s, Win95, Win/NT. Il inclue un assembleur 32 bits (avec une syntaxe pré-fixe ou post-fixe). Le traitement des macros est effectuée par le langage FORTH, mais le seul contexte utilisable en entrée ou en sortie est Win32For. Vous pouvez le trouver à l'adresse *ftp://ftp.forth.org/pub/Forth/win32for/*.

6.4.2 NASM

Il s'agit du *Netwide Assembler*, écrit en C, qui devrait être suffisement modulaire pour gérer toutes les syntaxes et les formats connus.

La version actuelle tourne bien, mais seulement sur de la syntaxe très simple, et pour générer uniquement du binaire. Aucun traitement de macros est intégré.

NASM évolue trop vite pour que ce document soit à jour. Toutefois, n'attendez pas à ce que cet assembleur implémente toutes les caractéristiques à moins que vous ne donniez un coup de main pour les implémenter. Consultez *http://www.dcs.warwick.ac.uk/~jules/nasm1.html*.

6.4.3 Tunes

Le projet OS Tunes développe son propre assembleur sous la forme d'une extension du langage Scheme, comme étant une partie de son processus de développement. Toutefois, il ne tourne pas d'une manière satisfaisante actuellement. Il est fort probable qu'un peu d'aide serait la bienvenue.

Cet assembleur manipule des arbres de syntaxe symboliques , donc il pourrait aussi bien servir pour un traducteur de syntaxe assembleur, un désassembleur, un compilateur/assembleur, etc. Voir *http://www.eleves.ens.fr:8080/home/rideau/Tunes/*.

6.4.4 Assembleurs non libres et non 32 bits pour x86.

Vous en saurez plus à ce sujet en lisant les bases de la programmation x86 dans la FAQ de Raymond Moon du forum Usenet `comp.lang.asm.x86`: *http://www2.dgsys.com/~raymoon/faq/asmfaq.zip*.

7 Méta-programmation et macro-programmation

La programmation en assembleur est casse-pieds, sauf pour des parties sensibles de programmes. Vous devrier utiliser les outils appropriés en fonction des outils que vous souhaitez créer. Donc ne choisissez pas l'assembleur lorsque cela ne correspond pas. C, OCAML, perl, Scheme, peuvent être un meilleur choix pour vos programmes.

Toutefois, il existe certains cas où ces outils ne donnent pas un contrôle assez fin sur votre machine, et l'assembleur est alors utile, voir nécessaire. Dans ces cas, vous allez apprécier un système de macro-programmation et de méta-programmation qui vous permet de ne définir qu'une seul fois des symboles que vous répettez et ainsi pouvoir les utiliser bon nombre de fois grâce à un système d'expansion. Cela permet une programmation plus sûre, de modifier le code plus simplement, etc. Un pur assembleur n'est généralement pas suffisant, même lorsque vous ne faîtes que de petites routines que vous utilisez dans des programmes C.

7.1 Qu'est-ce que cela veut dire?

7.1.1 gcc

`gcc` permet (et demande) à ce que vous spécifiez des contraintes sur les registres dans votre code en ligne, de telle manière que l'optimiseur le sache. Donc, l'assembleur en ligne est vraiment fait de modèles, et pas forcément de code au sens du terme.

Ensuite, vous pouvez transformer votre code en macros CPP, de telle manière que n'importe qui puisse s'en servir comme fonction en ligne ou macro C. Les fonctions en lignes ressembles de très près aux macros, mais elles sont parfois plus propres à utiliser. Faites attention car dans un tel cas, le code sera dupliqué, donc seules des étiquettes locales (comme `1:`) devraient être définies dans le code assembleur. Toutefois, une macro-commande pourrait autoriser le passage d'une étiquette non définie en local en paramètre. Notez également que certaines erreurs de votre code ou de gcc peuvent apparaître lorsque vous utilisez des fonctions en ligne en assembleur si les contraintes de registres n'ont pas été déclarées proprement.

Enfin, le langage C peut être considéré comme étant une bonne abstraction au langage assembleur, ce qui vous évite une bonne partie des problèmes de l'assembleur.

Prenez garde à certaines optimisations qui peuvent passer des arguments à des fonctions via des registres. Dans ce cas, ces fonctions pourraient ne plus être joignables de l'assembleur d'une manière standard, à moins que vous n'utilisiez le mot clef `asmlinkage`. Consultez les sources du noyau Linux pour plus d'exemples.

7.1.2 gas

`gas` ne possède absolument aucune capacité de gestion de macros. Toutefois, `gcc` fait passer les fichiers `.S` via `cpp` avant d'être donné à `gas`. Les fichiers `.s` ne sont générés qu'une seule fois et passés directement à `gas`. Encore une fois, consultez les sources de Linux.

7.1.3 gasp

Ce programme correspond au préprocesseur de macro commandes en assembleur de GAS. Consultez sa documentation en `texinfo`.

7.1.4 as86

Il gère quelques macros assez simple, mais je n'ai pas réussi à trouver de la documentation. Maintenant, les sources sont très parlants, donc si vous êtes intéressé, vous devriez les comprendre très facilement. Si vous avez besoin de plus que le stricte minimum, vous devriez utiliser un filtre externe (voir un peu plus bas...)

7.1.5 Autres assembleurs

- Win32FORTH : `CODE` et `END-CODE` sont des macros-commandes qui ne varient pas en fonction du mode, donc vous avez acccès à toute la puissance des mots clefs du langage FORTH dans le code assembleur ;
- NASM : aucune gestion des macros pour le moment. Consultez la partie sur les filtres externes.
- TUNES : il ne fonctionne pas encore mais le langage Scheme est un langage vraiment de haut niveau qui permet arbitrairement une programmation par macro-commandes.

7.2 Filtres externes

Quelque soit la gestion de macro supportée par votre assembleur, ou quelque soit le langage que vous utilisez (même C !), si le langage n'est pas assez puissant pour vous, vous pouvez passer les fichiers à travers un filtre externe en incluant une règle dans le Makefile, comme par exemple :

```
%.s:    %.S autre_dependances
        $(FILTRE) $(OPTIONS_FILTRE) < $< > $@
```

7.2.1 cpp

`cpp` n'est pas vraiment très expressif mais il est assez bon pour des opérations simples. Il est standard et appelé d'une manière transparente par `gcc`.

Comme exemple de limitation, vous ne pouvez pas déclarer d'objets de telle manière que les destructeurs soient automatiquement appelés à la fin du bloc de déclaration. Vous ne pouvez pas déclarer les données et le code qui va les manipuler, etc.

`cpp` est livré avec le compilateur C. `gcc` (voir ci-dessus) est un compilateur C libre, qui le contient.

7.2.2 m4

`m4` vous donne toute la puissante du traitement de macro instructions, avec un langage équivalent à celui de Turing, la récursivité, les expressions régulières, etc. Vous pouvez faire tout ce que vous ne pouviez pas faire avec `cpp`.

Récupérez *macro4th/This4th* à partir de *ftp://ftp.forth.org/pub/Forth/* dans `Reviewed/ ANS/ (?)`. Des exemples de programmation avancée sont disponibles dans les sources de `Tunes 0.0.0.25`.

Toutefois, il y a pas mal de problèmes de sémantiques avec les quotes qui vous obligent à utiliser des continuation explicites, des macros récursives si vous souhaitez faire de la programmation avancée (ce qui n'est pas sans rappeler – d'ailleurs, est-ce que quelqu'un a déjà essayé d'utiliser TeX comme un macroprocesseur pour autre chose que du traitement de texte ?). Ce n'est pas pire que `cpp` car il n'autorise pas les quotes et la récursivité.

La bonne version de `m4` a récupérer est la version GNU `m4 1.4` (ou plus récente si elle existe), qui possède le plus de caractéristiques et le moins de bugs et de limitations.

Macro-traitements avec vos propres filtres Vous pouvez écrire vos propres filtres d'expansion de macros avec les outils habituels : `perl`, `awk`, `sed`, etc. C'est rapide à faire, et vous contrôlez tout. Mais bien sûr, toute la puissante des traitement dépend de vous.

Macro-programmation Au lieu d'utiliser un filtre externe qui expanse les macro-commandes, une manière de travailler consiste a écrire des programmes qui écrivent des parties ou tous les autres programmes.

Par exemple, vous pourriez utiliser un programme qui génère le code source pour générer les tables correspondantes aux opérations sinus et cosinus, pour extraire une représentation code source d'un binaire, pour compiler vos images en routines d'affichage rapide, pour extraire de la documentation, du code d'initialisation et de terminaison, des tables de descriptions. Cela pourrait être un code source généré à partir de scripts perl, shell, scheme qui effectue le traitement (particulièrement utile lorsque certains types de données doivent être dupliqués dans bon nombre de tables de références croisées et dans du code).

Pensez-y !

8 Conventions d'appellation

8.1 Linux

8.1.1 Editions de liens avec gcc

C'est la solution la plus pratique.

Les arguments 32 bits sont empillés dans l'ordre inverse (de là, dépillés dans le bon ordre) au-dessus de l'adresse de retour 32 bits. Les registres `%ebp, %esi, %edi` et `%ebx` sont préservés. `%eax` contient le résultat, ou bien `%edx:%eax` dans le cas d'un résultat sur 64 bits.

Pile `FP` : je ne suis pas très sûr, mais je pense que son résultat se trouve dans `st(0)`, toute la pile d'appel étant conservée.

`gcc` possède certaines options pour modifier les conventions d'appels en réservant des registres, mettant des arguments dans des registres, ne pas gérer la FPU. Consultez les documents sur le i386.

Faites attention à bien déclarer une fonction `asmlinkage` si vous voulez que `gcc` utilise la convention standard (je ne sais pas trop ce qu'il fait avec les conventions modifiées).

8.1.2 ELF et a.out : problèmes rencontrés

Quelques compilateurs C rajoutent un blanc souligné avant chaque symbole, alors que d'autres ne le font pas.

En particulier, `gcc` a.out pour Linux effectue ce genre de rajout, alors que la version ELF ne le fait pas. Regardez comment le noyau gère cela (`linux/include/linux/linkage.h`).

Vous pouvez redéfinir la procédure de renomage C vers assembleur en insérant des expressions comme

```
void truc asm("zorglub") (void);
```

pour s'assurer que la fonction C `truc` sera appelée `zorglub` en assembleur.

Vous pouvez remarquer que l'utilitaire `objcopy` du paquetage `binutils` devrait vous permettre de transformer vos objets a.out en objets au format ELF, et peut-être même le contraire dans certains cas.

8.1.3 Appels systèmes directs sous Linux

Ce n'est pas vraiment recommandé car cela peut changer, et surtout parceque ce n'est pas portable. Enfin, ce n'est vraiment pas conseillé car la bibliothèque libc effectue certaines corrections ainsi que certaines extensions, comme par exemple `zlibc` (qui effectue de la décompression à la volée de fichiers compressés avec `gzip`). Par convention, il est recommandé d'utiliser la libc pour faire appelle à appels systèmes.

Maintenant, si vous ne voulez pas effectuer une édition de liens avec la libc, alors vous devriez regarder comment `linux-eforth-1.0c.tgz` le fait (*ftp://ftp.forth.org/pub/Forth/Linux/*). Les sources de Linux sont bien utiles, en particulier le fichier d'en-tête `asm/unistd.h`, qui décrit comment effectuer des appels systèmes...

D'une manière plus pratique, vous appellez l'intéruption $0x80, avec le nombre `__NR_syscallname` (voir `asm/unistd.h`) dans le registre `%eax`, et les paramètres (cinq maximum) se trouvent respectivement dans les registres `%ebx, %ecx, %edx, %esi, %edi`. La valeur de retour se trouve dans le registre `%eax`, et si le nombre est négatif, alors il s'agit d'une erreur que la libc va transformer en `errno`. La pile utilisateur n'est pas modifiée, donc ce n'est pas la peine d'en avoir une valide lors de l'appel système.

8.2 DOS

Bon nombre d'extensions DOS sont fournies avec des services DOS. Lisez leur documentation, mais bien souvent, ils ne font que simuler l'interruption 0x21 et ainsi de suite, comme si vous étiez en mode réel (je doute que cela marche avec des opérateur 32 bits en appellant des services du DOS en 16 bits).

La documentation concernant DPMI ou autres se trouve sur *ftp://x2ftp.oulu.fi/pub/msdos/programming/*.

DJGPP est fournis avec sa propre libc (limitée).

Il est possible d'effecter une cross-compilation de Linux vers Dos, mais les patches existants ont été écrit pour la version a.out de `gcc`. Peut-être que les dernières versions ELF du compilateur n'ont pas besoin d'être patchées : je n'en sais rien.

8.3 Votre propre système d'exploitation

C'estcedontbeaucoupdedveloppeursenassembleurparlent

8.3.1 Code du *Boot loader* et passage en mode 32 bits

8.3.2 Les bases de la protection

8.3.3 Gestion des interruptions

8.3.4 Modes V86/R86 pour l'utilisation des services du système 16 bits

8.3.5 Où trouver des informations sur tout ça ?

Rajoutezdespointeurssurd'autresdocumentsdanscettesection!

La source principale d'information est les sources des systèmes d'exploitations existant. Bon nombre de pointeurs se trouve sur la page WWW suivante : *http://www.eleves.ens.fr:8080/home/rideau/Tunes/Review/OSes.html*.

9 A faire

- remplir les sections incomplètes
- ajouter plus de liens sur les programmes...

- ajouter des exemples simples pour illustrer la syntaxe, la puissance et les limitations de chaque solution proposée.