

FAQ Assembleur

Date de publication : 08/11/2004

Dernière mise à jour : 29/04/2007

Cette F.A.Q. a été réalisée à partir des questions fréquemment posées sur le forum **Assembleur** de <http://www.developpez.com> et de l'expérience personnelle des auteurs. Nous tenons à souligner que cette F.A.Q. ne garantit en aucun cas que les informations qu'elle propose soient correctes. Les auteurs font le maximum, mais l'erreur est humaine. Cette F.A.Q. ne prétend pas non plus être complète. Si vous trouvez une erreur, ou que vous souhaitez devenir rédacteur, lisez Comment participer à cette F.A.Q. ?.

Nous espérons que cette F.A.Q. saura répondre à un maximum de vos questions. Nous vous souhaitons une bonne lecture.

L'équipe Assembleur de Developpez.com.

Ont contribué à cette FAQ :

Clément Cunin - Gysmo - le mage tophinus - Neitsa
(<http://neitsa.developpez.com/>) - Romain Tartièrè
(Smortex) (<http://smortex.developpez.com/>) - TangiX
- VBurel - Alcatîz (<http://Alcatîz.developpez.com/>) -

1. Information générale (3)	4
2. F.A.Q. Générale (22)	6
2.1. Technologies (3)	8
2.2. Terminologie (6)	9
2.3. Compilation (4)	11
2.4. Désassemblage et reverse engineering (1)	12
2.5. Emulation (2)	13
3. F.A.Q. Assembleur x86 (11)	14
3.1. Matériel (3)	16
3.2. Interruptions (2)	18
3.3. Programmation Win32 (1)	19
3.4. Assembleur et Delphi (1)	20
4. F.A.Q. Assembleur x86 / 64 (14)	21
4.1. Généralités (6)	22
4.2. Questions spécifiques à la programmation (8)	24

[Sommaire > Information générale](#)**Comment bien utiliser cette F.A.Q. ?****Auteurs : Clément Cunin ,****Le but :**

Cette F.A.Q. a été conçue pour être la plus simple possible d'utilisation. Elle tente d'apporter des réponses simples et complètes aux questions auxquelles sont confrontés tous les débutants (et les autres).

- **L'organisation :**

Les questions sont organisées par thème, les thèmes pouvant eux-mêmes contenir des sous-thèmes. Lorsqu'une question porte sur plusieurs thèmes, celle-ci est insérée dans chacun des thèmes, rendant la recherche plus facile.

- **Les réponses :**

Les réponses contiennent des explications et des codes sources. Certaines sont complétées de fichiers à télécharger contenant un programme de démonstration. Ces programmes sont volontairement très simples afin qu'il soit aisé de localiser le code intéressant. Les réponses peuvent également être complétées de liens vers d'autres réponses ou vers un autre site en rapport.

- **La recherche :**

Un moteur de recherche sur les pages de la F.A.Q. est disponible, utilisez le bouton [Rechercher] situé en haut de page.

- **Nouveautés et mises à jour :**

Lors de l'ajout ou de la modification d'une question/réponse, un indicateur est placé à côté du titre de la question. Cet indicateur reste visible pour une durée de 15 jours afin de vous permettre de voir rapidement les modifications apportées.

Nous espérons que cette F.A.Q. pourra répondre à vos questions.

lien : [Comment participer à cette F.A.Q. ?](#)**Comment participer à cette F.A.Q. ?****Auteurs : Clément Cunin ,**

Cette F.A.Q. est ouverte à toute collaboration. Pour éviter la multiplication des versions, il serait préférable que toutes collaborations soient transmises aux administrateurs de la F.A.Q.

Plusieurs compétences sont actuellement recherchées pour améliorer cette F.A.Q. :

- **Rédacteur :**

Bien évidemment, toute nouvelle question/réponse est la bienvenue.

- **Correcteur :**

Malgré nos efforts des fautes d'orthographe ou de grammaire peuvent subsister. Merci de contacter les administrateurs si vous en débusquez une... Idem pour les liens erronés.

Remerciements

Auteurs : Gysmo ,

Remerciements à tous ceux qui ont contribué de près ou de loin à l'élaboration de cette F.A.Q.

Remerciements à l'équipe des rédacteurs du forum Assembleur, qui ont grandement contribué à la réalisation de cette F.A.Q.

Remerciements aux correcteurs, qui nous ont bien aidés à corriger les fautes de français (Piotrek tout seul en fait...).

Enfin, un grand merci à Clément Cunin, auteur originel du cadre de cette F.A.Q. et auteur de la F.A.Q. Java.

[Sommaire > F.A.Q. Générale](#)

Qu'est ce que l'Assembleur ?

Auteurs : Romain Tartière (Smortex) ,

L'Assembleur est le langage de programmation de plus bas niveau. Cela signifie qu'il est très proche du matériel. A chaque instruction correspond un code machine, qui peut être compris par le microprocesseur. Le code assembleur est donc la version lisible du code machine.

Pourquoi programmer en Assembleur ?

Auteurs : Romain Tartière (Smortex) ,

S'il est de nos jours impensable de programmer entièrement une application en Assembleur (sauf à des fins d'apprentissage bien sûr), nombreuses situations font que l'on doit avoir recours à ce langage. C'est par exemple le cas lorsque la vitesse est un facteur critique, et que la moindre microseconde est précieuse, ou encore pour écrire un gestionnaire de démarrage (Boot-Loader), etc...

Que programmer en Assembleur ?

Auteurs : Romain Tartière (Smortex) ,

Tout ce qui doit être optimisé au niveau de la taille (si on a que 512 octets de mémoire par exemple, cas de certains microcontrôleurs), de la vitesse d'exécution, ou ne peut pas être fait avec autre chose que de l'Assembleur (par exemple un driver de disque dur).

Comment intégrer de l'assembleur dans un langage évolué ?

Auteurs : Romain Tartière (Smortex) ,

La plupart des langages évolués permettent d'utiliser l'Assembleur dans les programmes. La syntaxe à utiliser est dépendante du langage et du compilateur utilisé.

Par exemple, sous Delphi (et plus généralement, en Pascal) :

```
asm
{ code en assembleur }
end;
```

En C :

```
asm {
/* Code en assembleur */
```

}

Comment connaître le temps d'exécution de mes fonctions ?

Auteurs : Romain Tartière (Smortex) ,

Il y a deux méthodes. La première, qui fonctionne dans tous les cas, est de compter le nombre de cycle d'horloge nécessaire à l'exécution de chaque instruction (Dans la documentation du microprocesseur / microcontrôleur). Une fois que l'on connaît le nombre de cycles d'horloge nécessaires, il suffit de le multiplier par la période d'un cycle d'horloge interne (Qui est généralement différente de la période de l'horloge du composant).

Par exemple, sur un 68HC11 (Un microcontrôleur de Motorola), on a :

```
LDAA #$FF ; 2 cycles
STAA PORTB ; 2 cycles
```

$2+2 = 4$

Quartz à 20 MHz, fréquence interne : $20 / 4 = 5\text{MHz}$

Période : $1/(5*10^6) = 500*10^{-9} = 500\text{ns}$

$4 * 500 = 2\mu\text{s}$

La seconde méthode n'est utilisable que si elle est supportée par votre micro. Elle consiste à utiliser les instructions spécifiques à votre micro permettant de mesurer les durées d'exécution.

Comment échanger la valeur de deux registres sans utiliser d'espace d'échange ?

Auteurs : Romain Tartière (Smortex) ,

Il faut utiliser la fonction logique XOR. En effet, Si on a deux registres A et B dont on souhaite échanger les données, et que l'on fait :

```
A <- A XOR B
B <- B XOR A
A <- A XOR B
```

On remarque que A contient la donnée initiale de B et B celle de A (Bien sûr, ça n'est pas utile si le micro possède une instruction pour le faire... Mais sur un petit micro RISC, si on est à un octet près, c'est utile).

Par exemple, sur x86, pour échanger ax et bx :

```
xor ax,bx
xor bx,ax
xor ax,bx
```

Sommaire > F.A.Q. Générale > Technologies

Quelle est la différence entre un microprocesseur et un micro-contrôleur ?

Auteurs : Romain Tartière (Smortex) ,

Un microprocesseur est un circuit électronique permettant de faire divers calculs. Il nécessite d'autres circuits pour fonctionner, tels que RAM, ROM, et périphériques d'entrées/sorties. La communication entre ces éléments se fait grâce à plusieurs bus : le bus d'adresse qui permet choisir "l'endroit" où lire et écrire les données, le bus de donnée qui fait transiter les données, et le bus de contrôle qui renseigne sur l'état du micro (Débordement, Interruptions activées, etc ...). Un microcontrôleur possède tous les éléments dans un même boîtier. Il suffit donc de le relier aux périphériques pour le mettre en oeuvre.

Quelle est la différence entre le "RISC" et le "CISC" ?

Auteurs : Romain Tartière (Smortex) ,

Il existe deux familles de microprocesseurs / microcontrôleurs (micros). Les micros CISC (Complete Instruction Set Computers) possèdent un grand nombre d'instructions (plusieurs dizaines, voir centaines) permettant de réaliser des opérations très variées. Chaque instruction est exécutée en un certain nombre de cycles d'horloge, dépendant de l'instruction elle même, mais aussi (et surtout) du mode d'adressage utilisé. L'autre famille de micros, RISC (Reduced Instruction Set Computers) possède un nombre d'instructions bien plus faible (quelques dizaines au maximum) qui permettent de réaliser des opérations simples (opérations logiques, sommes, etc...). Mais chaque instruction est exécutée en un seul cycle d'horloge. Il en résulte une vitesse de calcul très supérieure à celle de la famille CISC.

Les circuits tels que les x86, 68HC11 sont des circuits CISC, les microprocesseurs de Mac, les Pic, les processeurs Alpha sont du type RISC.

Quelle est la différence entre une architecture de "Von Neumann" et de "Harvard" ?

Auteurs : Romain Tartière (Smortex) ,

Dans une architecture de type Von Neumann, les codes opérations (opcodes) et les paramètres sont stockés les uns derrière les autres dans le programme. Avec une architecture Harvard, les opcodes et leurs paramètres sont situés à des endroits différents.

Les x86, 68HC11, ont une architecture Von Neumann. Les Pic ont une architecture Harvard.

Sommaire > F.A.Q. Générale > Terminologie

Comment est organisée la mémoire ?

Auteurs : Romain Tartière (Smortex) ,

La mémoire est composée d'une suite d'emplacements numérotés séquentiellement, et pouvant contenir un nombre codé sur 8 bits (Donc compris entre 0 et 255). Chaque emplacement peut être lu ou écrit ; pour cela, il faut connaître son adresse.

Qu'est ce qu'une Interruption ?

Auteurs : Romain Tartière (Smortex) ,

Une interruption est un programme qui est exécuté lorsqu'un évènement particulier se produit. On distingue deux types d'interruption : les interruptions matérielles, qui sont déclenchées lorsqu'un évènement physique se produit (On appuie sur une touche du clavier par exemple) et les interruptions logicielles, qui sont appelées directement par le programme.

Qu'est-ce qu'une adresse ?

Auteurs : Romain Tartière (Smortex) ,

Il s'agit de l'endroit où est stockée une donnée. Selon le micro utilisé, elle peut être un nombre qu'on appelle adresse, ou deux nombres associés : le segment et l'offset, que l'on note segment:offset.

lien : [Qu'est-ce qu'un "segment" ?](#)

lien : [Qu'est ce qu'un "offset" ?](#)

Qu'est-ce qu'un "segment" ?

Auteurs : Romain Tartière (Smortex) ,

Si on imagine que la mémoire est une route, les bornes kilométriques représentent les segments. Cela signifie que la mémoire est découpée en morceaux, et que chaque morceau contient un certain nombre d'informations.

Qu'est ce qu'un "offset" ?

Auteurs : Romain Tartière (Smortex) ,

En anglais, "offset" signifie "décalage". La mémoire est décomposée en segments; pour localiser une donnée précise, il faut savoir dans quel segment elle se trouve, et le décalage qu'il y a entre cette donnée et la première donnée du segment. C'est ce qu'on appelle offset.

Qu'est ce qu'un "registre" ?

Auteurs : Romain Tartière (Smortex) ,

Il s'agit d'une zone particulière du micro qui contient un certain nombre de bits (généralement 8 pour les petits microcontrôleurs, 32 voire 64 pour les PC, et encore plus pour d'autres systèmes, 128 pour une PlayStation II par exemple). Les données dans ces emplacements sont celles traitées par le micro. Il peut réaliser plusieurs actions sur ses

registres : addition, soustraction, décalage, etc. Chaque instruction en assembleur dépend de, et modifie le contenu des registres du micro.

Sommaire > F.A.Q. Générale > Compilation

Pourquoi mon compilateur ne démarre pas ?

Auteurs : **TangiX** ,

La première chose à laquelle s'attend un débutant est de lancer une EDI; or les compilateurs assembleur sont le plus souvent des programmes DOS (ou console WIN32) qui demandent des paramètres pour s'exécuter.

Exemple : `nasm mon_prog.asm -o mon_prog.exe`

Il est toutefois possible de se servir d'une IDE qui automatise la compilation.

Quels sont les compilateurs pour l'Assembleur ?

Auteurs : **Neitsa** ,

- **Masm** : Le site officiel du package MASM32. L'assembleur pour Windows et ses APIs principalement.
- **Nasm** : Le site officiel de l'assembleur NASM. (Intel x86 code. Plateformes MSDOS, Win32, Linux).
- **Fasm** : Site officiel de FASM. (Assembleur Intel x86 pour MSDOS, Win32, Linux).
- **Tasm** : Site officiel de Borland Turbo Assembler (plateformes X86 - Dos et Windows).
- **HLA (High level assembly)** : Un assembleur avec syntaxe et fonctions de haut niveau. Par le créateur de l'AOA.
- **GoAsm** : Un assembleur uniquement pour Windows et sa programmation.
- **RosAsm** : Site officiel de RosAsm. (Windows et Reactos).
- **Yasm** : Dos / Win32 / Linux (support 64bits pour le format ELF).
- **LZasm (Lazy Assembler)** : Dos et Windows sous X86.
- **Gas (GNU Assembler)** : Assembleur multi plateformes.
- **NBAsm (New Basic Assembler)** : MS-DOS seulement.

Quels sont les linkers pour l'assembleur ?

Auteurs : **Neitsa** ,

- **alink** : Très bon éditeur de liens (16/32 bits).
- **Qlink** : Éditeur de liens 16 bits.

Quels sont les Environnements de Développement (EDI) pour l'assembleur ?

Auteurs : **Neitsa** ,

- **AsmEditor** : Très bon IDE, multi-support de compilateurs, facilité d'utilisation et ergonomie intuitive.
- **RadASM** : Très bon IDE pour l'asm. Supporte masm / tasm / fasm / nasm / goasm / hla.
- **Winasm studio** : Un concurrent sérieux aux IDE cités ci-dessus.

Sommaire > F.A.Q. Générale > Désassemblage et reverse engineering

Désassemblage et reverse engineering... Qu'est-ce que c'est ?

Auteurs : Romain Tartière (Smortex) ,

Le désassemblage est une technique qui consiste à retrouver le code source d'un programme compilé. Cela revient à faire l'inverse de ce que fait le compilateur, soit transformer les opcodes en mnémoniques. S'il peut être envisagé de reconstituer les sources d'un petit programme, cela devient vite irréalisable pour un programme plus important. En effet, le désassemblage ne permet que de retrouver le code source Assembleur, et pas toutes les informations qui permettent sa compréhension (Noms des étiquettes, commentaires, ...).

Sommaire > F.A.Q. Générale > Emulation

Qu'est-ce qu'un émulateur ?

Auteurs : Romain Tartière (Smortex) ,

Un émulateur est un programme capable de simuler le fonctionnement d'un PC au sein d'un autre PC. Cela permet par exemple de faire tourner un système d'exploitation sur un autre système (Par exemple, faire fonctionner Windows sous Linux). Il existe deux grandes familles d'émulateurs :

- **La première transmet les instructions au microprocesseur du PC, qui les exécute. La machine émulée possède donc le même processeur que le PC hôte, ce qui peut être un handicap.**
- **Il existe donc également des émulateurs qui simulent aussi le microprocesseur. On peut donc tester le comportement d'un programme sur une machine 64 bits, même si l'on en a pas a disposition. Ce second type d'émulateur présente l'inconvénient d'être assez lent.**

Quand utiliser un émulateur ?

Auteurs : Romain Tartière (Smortex) ,

On a généralement recours à un émulateur dans trois cas :

- **Lorsque l'on développe un système d'exploitation, et qu'on ne désire pas rebooter a chaque compilation du code pour vérifier le fonctionnement du programme;**
- **Lorsque l'on veut avoir de nombreuses informations sur l'état du système lors de l'exécution d'un programme;**
- **Lorsque l'on souhaite tester le fonctionnement d'un programme sur une machine équipée d'un autre microprocesseur.**

[Sommaire](#) > F.A.Q. Assembleur x86

Qu'est-ce qu'un x86 ?

Auteurs : Romain Tartière (Smortex) ,

x86 fait référence a tous les microprocesseurs de PC, depuis le 8086. Cela inclut donc (Pour les x86 les plus connus) : les 8080, 286, 386, 486, 586, 686. Le numéro peut être précédé d'un 'i' : i586. Il s'agit d'un problème d'enregistrement des marques : elles doivent nécessairement débiter par une lettre.

Où trouver des informations sur le microprocesseur ?

Auteurs : Romain Tartière (Smortex) ,

Il faut télécharger les DataSheets fournies par le fabricant. A cet effet, on se rendra sur le site <http://www.intel.fr> ou <http://www.amd.com> (Ou autre selon le microprocesseur que l'on possède).

C'est quoi Ring 0, Ring 1, etc... ?

Auteurs : Romain Tartière (Smortex) ,

Le "Ring" désigne le niveau de privilège. Il en existe quatre : de Ring 0 à Ring 3. Ils ont été créés afin de hiérarchiser les tâches du microprocesseur.

Le niveau de privilège le plus élevé est le niveau 0; il correspond au fonctionnement du Kernel du système d'exploitation. Les niveaux 1 et 2 au système d'exploitation lui même (drivers, services, etc...) et le niveau 3 aux applications de l'utilisateur.

Quels sont les différents modes d'adressage d'une donnée ?

Auteurs : Alcatiz ,

- **Adressage direct :** l'adresse de la donnée fait partie de l'instruction :

```
mov ax,Truc
mov ax,ds:[0020h]
```

- **Adressage immédiat :** en réalité, ce n'est pas un adressage puisque c'est la donnée elle-même qui fait partie de l'instruction :

```
mov ax,20
```

- **Adressage indirect :** on met dans l'instruction un ou plusieurs élément(s) qui contie(nne)nt l'adresse de la donnée :

```
mov ax,[bx]
```

Attention ! Seuls les registres suivants peuvent contenir une adresse :

bx (registre de base dans le segment de données);

bp (registre de base dans la pile);

si et **di** (registres d'index dans le segment de données).

L'instruction suivante est une erreur de syntaxe :

```
mov ax, [cx]
```

Il existe différents adressages indirects :

Basé ou indexé simple (avec éventuellement ajout d'un offset) :

```
mov ax, Truc [bx]  
mov ax, [bp]  
mov ax, [si]  
mov ax, [di]
```

Basé-indexé (avec éventuellement ajout d'un offset) :

```
mov ax, [bp][di]  
mov ax, Truc [bx][si]
```

Ce type d'adressage est très utile pour gérer des tableaux.

Sommaire > F.A.Q. Assembleur x86 > Matériel

Comment arrêter et redémarrer le PC ?

Auteurs : Romain Tartière (Smortex) ,

Il existe plusieurs méthodes. La plus simple est d'utiliser l'interruption 19h, mais cela présente de nombreux inconvénients, du fait que la mémoire n'est pas effacée et que les vecteurs d'interruption ne sont pas réinitialisés pour la zone [00h..1Ch]. On a donc des chances de causer un plantage au lieu de redémarrer le système. On préférera donc à cette première méthode la suivante : écrire 1234h en 0040:0072 suivi d'un saut long en FFFF:0000, ce qui a pour effet de causer un redémarrage à chaud. Enfin, si on est en mode protégé, on peut redémarrer en causant une "triple fault", ce qui se fait très facilement en altérant le registre IDTR et en exécutant une interruption logicielle.

Comment s'écrit un boot-loader de base ?

Auteurs : Romain Tartière (Smortex) , VBurel , le mage tophinus ,

Un boot-loader est un programme de 512 octet qui se trouve sur le premier secteur d'un disque et est exécuté par le BIOS dans le but de charger le système d'exploitation. Il est copié à l'adresse 07C0:0000 avant d'être exécuté. Si le programme doit faire plus de 512 octets, le boot-loader aura le rôle de charger les secteurs suivants en mémoire. Un secteur de boot minimaliste est tel que celui-ci (Écrit avec FASM) :

```

    Org      0;

    Jmp     07C0h:Start

;-----;
; Programme principal ;
;-----;
Start:
    Mov     Ax,Cs
    Mov     Ds,Ax
    Mov     Es,Ax

; Programme de boot proprement dit...
Hang:
    Jmp    Hang

End.:

;--- 00 jusqu'à 510 ---
times 510 - End. db 0
dw 0AA55h
  
```

On peut également procéder de cette façon :

```

    Org      0;

    Jmp     Start

;-----;
; Programme principal ;
;-----;
Start:
    Cli
    Mov     Ax,07C0h
    Mov     Ds,Ax
    Mov     Es,Ax
    ; Initialisation de la pile ici
  
```

```
Sti
; Programme de boot proprement dit...
Hang:
    Jmp Hang
End.:
;--- 00 jusqu'à 510 ---
times 510
dw 0AA55h
```

L'utilisation de Cli et Sti dans le second exemple est recommandée lors de l'initialisation de SS et SP. Pour mettre le mot magique en fin de secteur, vous pouvez utiliser ce code pour TASM (et MASM) :

```
fin_prog:
    times 510 - fin_prog db 0
dw 0AA55h
```

... et pour NASM :

```
times 510-($-$$) db 0
dw 0AA55h
```

Comment écrire sur le port parallèle ?

Auteurs : Romain Tartière (Smortex) ,

L'instruction :

```
out 378h, ax
```

ne fonctionne pas... et c'est normal ! Si vous regardez la documentation de votre micro, vous noterez que le numéro du port doit impérativement appartenir à [0..255]. La solution est guère plus loin dans la documentation : il faut passer par le registre DX. On peut alors écrire dans un plus grand nombre de ports : [0..1023].

```
mov dx, 378h
out dx, ax
```

Sommaire > F.A.Q. Assembleur x86 > Interruptions

Comment créer ses propres interruptions ?

Auteurs : Romain Tartière (Smortex) ,

La première étape consiste à écrire le code de l'interruption lui même. Il est identique à celui de l'écriture d'une procédure "normale" si ce n'est qu'il se termine par une instruction iret au lieu de ret.

Par exemple, on va créer une interruption FDh :

```
IntFD:  
[...]  
iret
```

La seconde étape consiste à initialiser le vecteur d'interruption de manière à pointer la zone mémoire qui contient le programme à exécuter.

Chaque vecteur occupe 4 octets à partir de l'adresse 0000:0000 et qui indiquent la zone de la mémoire où se trouve le code de l'interruption. C'est donc cet espace qu'il faut mettre à jour.

Exemple :

```
Init_IntFD:  
push ax  
push es  
xor ax,ax  
mov es,ax; Es <--- 0  
mov word[es:0FDh*4],IntFD; Initialisation du vecteur  
mov word[es:0FDh*4+2],cs; de l'interruption  
pop es  
pop ax  
ret
```

C'est terminé ! Il ne reste plus qu'à déclencher l'interruption :

```
int 0FDh
```

Où trouver la liste des interruptions ?

Auteurs : Romain Tartière (Smortex) ,

Pour programmer sur un système x86, il est indispensable d'avoir une table des interruptions sous la main... On peut en trouver de nombreuses plus ou moins complètes sur le Web... Celle de Ralf BROWN est très complète. Vous la trouverez en consultation en ligne ici : <http://www.ctyme.com/rbrown.htm> Ou si vous préférez tout télécharger, rendez-vous ici : <http://www-2.cs.cmu.edu/afs/cs/user/ralf/pub/WWW/files.html>

Sommaire > F.A.Q. Assembleur x86 > Programmation Win32

Comment appeler les API Windows ?

Auteurs : Romain Tartière (Smortex) ,

De nombreux compilateurs proposent des fonctionnalités permettant un appel simple des API Windows. En général il s'agit d'une macro "invoke" qui empile les paramètres et déclenche la fonction demandée.

Exemple (MASM) :

```
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
[...]
INVOKE MessageBox, NULL, ADDR msgbox_caption, ADDR appname, MB_OK
INVOKE ExitProcess, NULL
```

Sommaire > F.A.Q. Assembleur x86 > Assembleur et Delphi

Comment lier des fichiers OBJ à un programme Turbo Pascal / Borland Pascal / Delphi ?

Auteurs : Alcafiz ,

Il est possible de faire cela en déclarant la procédure ou fonction comme ceci :

```
{ $L MonObj.obj }  
procedure MaProc (x, y : Integer); external;
```

Sommaire > F.A.Q. Assembleur x86 / 64

[Sommaire](#) > [F.A.Q. Assembleur x86 / 64](#) > Généralités

Où trouver de la documentation pour la programmation x64 en Assembleur ?

Auteurs : Neitsa ,

- Intel : <http://www.intel.com/products/processor/manuals/index.htm>
- AMD : http://www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_875_7044,00.html
- AMD64 ABI (Application binary interface): <http://www.x86-64.org/documentation/abi.pdf> (concerne aussi les processeurs Intel, mais vise surtout Linux)
- x64 Software Convention (MSDN) : <http://msdn2.microsoft.com/en-us/library/7kcdt6fy.aspx>
- Gentle Introduction to x86-64 Assembly (Jan Hubicka) : <http://www.x86-64.org/documentation/assembly.html>
- Writing 64-bit programs (Jeremy Gordon) : <http://www.jorgon.freemove.co.uk/GoasmHelp/64bits.htm>
- Under the hood, programming for 64-bit Windows (Matt Pietrek) : <http://msdn.microsoft.com/msdnmag/issues/1100/hood/>

J'ai un processeur 64 bits, puis-je exécuter des programmes ou binaires 64 bits ?

Auteurs : Neitsa ,

Bien sûr, mais à une seule condition : si votre système d'exploitation est un système 64 bits !

Les processeurs 64 bits peuvent exécuter du code 16, 32 ou 64 bits mais c'est le système d'exploitation qui détermine dans quel mode d'exécution il peut exécuter du code. Cela implique qu'un système d'exploitation 32 bits fonctionnant sur un processeur 64 bits ne pourra pas exécuter du code 64 bits.

N.B : Windows Vista 64 bits n'exécute plus les programmes 16 bits. Les programmes 32 bits, eux, sont exécutés au travers de WOW64.

Y a-t-il de nouvelles instructions avec les processeurs x86 64 bits ?

Auteurs : Neitsa ,

Une seule nouvelle instruction d'ordre "général" a fait son apparition : MOVSSD. Certaines des instructions sont simplement étendues pour manipuler des données 64 bits tandis que d'autres ne sont plus disponibles dans le mode 64 bits.

Les processeurs de dernières générations apportent toutefois des instructions spécialisées, notamment au travers des jeux d'instructions SSE3 et SSSE3 et des instructions de virtualisation, mais ces instructions ne sont pas réservées spécifiquement au mode d'exécution 64 bits.

Les registres généraux sont-ils toujours les mêmes que sous 32 bits ?

Auteurs : Neitsa ,

On trouve toujours les mêmes registres généraux que sous 32 et 16 bits. Toutefois les registres ont été étendus à 64 bits. Pour prendre un exemple avec EAX (qui, étendu à 64 bits, devient RAX) :

En mode 64 bits, on peut toujours accéder à :

- EAX : partie basse 32 bits (bits 0 à 31) de RAX.
- AX : partie basse 16 bits (bits 0 à 15) de EAX.
- AH : partie haute 8 bits (bits 8 à 15) de AX.
- AL : partie basse 8 bits (bit 0 à 7) de AX.

On ne peut ni accéder directement à la partie haute 16 bits de EAX, ni à la partie haute 32 bits de RAX.

Quels sont les nouveaux registres généraux des processeurs x86 64 bits ?

Auteurs : Neitsa ,

Outre les registres 32 bits étendus à 64 bits, on trouve 8 nouveaux registres généraux. Les nouveaux registres généraux 64 bits, au nombre de 8, sont nommés de R8 à R15 et sont utilisables tout comme les autres registres généraux. On dénombre donc à présent les registres suivants :

- 16 registres 8 bits « bas » : AL, BL, CL, DL, SIL, DIL, BPL, SPL, R8B, R9B, R10B, R11B, R12B, R13B, R14B, R15B. [Avec #B' pour BYTE]. On notera ici que les parties 8 bits de RSI (SIL), RDI (DIL), RBP (BPL) et RSP (SPL) sont désormais accessibles directement (ce qui n'était pas le cas en 16 ou 32 bits où, par exemple, la partie 8 bits du registre ESI n'était pas accessible directement).
- 4 registres 8 bits « haut » : AH, BH, CH, DH (utilisable seulement dans un cas bien précis : quand un préfixe REX est présent).
- 16 registres 16 bits : AX, BX, CX, DX, DI, SI, BP, SP, R8W, R9W, R10W, R11W, R12W, R13W, R14W, R15W. [Avec #W' pour WORD]
- 16 registres 32 bits : EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D, R9D, R10D, R11D, R12D, R13D, R14D, R15D. [Avec #D' pour DWORD].
- 16 registres 64 bits : RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8, R9, R10, R11, R12, R13, R14, R15.

Quels sont les nouveaux registres autres que les registres généraux ?

Auteurs : Neitsa ,

On notera l'apparition ou l'extension des registres suivants :

- 8 nouveaux registres SSE : XMM8 à XMM15.
- Le passage d'EFLAG (registre de drapeaux) en 64 bits, qui devient : RFLAG.
- Le pointeur d'instruction (EIP en 32 bits) passe lui aussi en 64 bits et devient : RIP.
- 8 nouveaux registres système de contrôle : Cr8 à Cr15.
- 8 nouveaux registres de déboguage : Dr8 à Dr15.

Sommaire > F.A.Q. Assembleur x86 / 64 > Questions spécifiques à la programmation

Quels compilateurs pour la programmation x86 64 bits ?

Auteurs : Neitsa ,

Windows :

- **GoAsm** : <http://www.jorgon.freemove.co.uk/>
- **MASM (64)** : fait partie du SDK de Windows (exécutable ml64.exe)

Cross plate-forme (au minimum Linux & Windows) :

- **FASM** : <http://flatassembler.net/>
- **YASM** : <http://www.tortall.net/projects/yasm/> (syntaxe NASM)
- **Gas** : (au travers de GCC par exemple) <http://www.gnu.org/software/binutils/>

(merci à Gamera pour le lien vers FASM)

Visual C++ déclenche une erreur lorsque j'utilise le mot clé `_asm`, pourquoi ?

Auteurs : Neitsa ,

Visual C++ ne permet plus l'utilisation de l'inlining Assembleur (Assembleur incorporé dans un programme en C ou C++).

Pour pallier à ce problème vous pouvez :

- Utiliser les intrinsics pour x64 : [http://msdn2.microsoft.com/en-us/library/26td21ds\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/26td21ds(VS.80).aspx)
- Utiliser un fichier `.asm` séparé du code en langage de haut niveau.

Quelle est la convention d'appel pour les processeurs x86 64 bits ?

Auteurs : Neitsa ,

Il n'existe plus qu'une seule convention d'appel en mode 64 bits. Cette dernière est semblable au type d'appel `_fastcall`, ce qui implique que les arguments, lors d'un appel de fonction, sont d'abord passés au travers des registres.

Pour Windows :

- Lorsqu'il y a 4 arguments entiers (ou moins), ceux-ci sont passés par les registres suivants, dans cet ordre précis : **RCX, RDX, R8 et R9**.
- Lorsqu'il y a plus de 4 arguments entiers, ceux-ci sont poussés sur la pile.
- Les arguments à virgule flottante (double ou float) sont pris en charge par les registres XMM (dans cet ordre précis) : **XMM0, XMM1, XMM2, XMM3**.
- Les arguments supérieurs à 64 bits (par ex. les doubles quadword [128 bits]) poussent leurs adresses sur la pile.

Ref. : [http://msdn2.microsoft.com/en-us/library/ms235286\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/ms235286(VS.80).aspx)

Ref. : [http://msdn2.microsoft.com/en-us/library/zthk2dkh\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/zthk2dkh(VS.80).aspx)

Ref. : <https://www.microsoft.com/france/msdn/visualc/introduction-convention-appel-64bits.mspx>

Pour Linux :

- Lorsqu'il y a 6 arguments entiers (ou moins), ceux-ci sont passés par les registres suivants, dans cet ordre précis : **RDI, RSI, RDX, RCX, R8, R9**.

- Lorsqu'il y a plus de 6 arguments entiers, ceux-ci sont poussés sur la pile.
- Les arguments à virgule flottante (double ou float) sont pris en charge par les registres XMM (dans cet ordre précis) : XMM0 à XMM7.
- Les arguments supérieurs à 64 bits (par ex. les doubles quadword [128 bits]) poussent leurs adresses sur la pile.

Ref. : AMD64 ABI.

Quels sont les registres utilisés pour les retours de fonctions ?

Auteurs : Neitsa ,

Pour Windows :

La valeur renvoyée par une fonction est placée dans le registre RAX, à moins que le résultat soit un type en virgule flottante, qui est alors renvoyé dans XMM0.

Pour Linux :

La valeur renvoyée par une fonction est placée dans le registre RAX et/ou RDX, à moins que le résultat soit un type en virgule flottante, qui est alors renvoyé dans XMM0 et/ou XMM1. Le retour de fonction peut aussi s'effectuer sous ST(0) ou ST(1) si la fonction manipule des données via le coprocesseur mathématique à virgule flottante (x87).

Quels sont les registres préservés ou détruits lors d'un appel de fonction ?

Auteurs : Neitsa ,

Pour Windows :

- D'un appel à l'autre, les registres suivants doivent être préservés : RBX, RBP, RDI, RSI, R12 à R15, XMM6 à XMM15.
- Les registres suivants sont volatiles et peuvent donc être détruits dans la fonction appelée : RAX, RCX, RDX, R8 à R11, ST(0) à ST(7), XMM0 à XMM5.

Ref. : [http://msdn2.microsoft.com/en-us/library/9z1stfyw\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/9z1stfyw(VS.80).aspx)

Ref. : [http://msdn2.microsoft.com/en-us/library/6t169e9c\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/6t169e9c(VS.80).aspx)

Pour Linux :

- D'un appel à l'autre, les registres suivants doivent être préservés : RBX, RBP, R12 à R15.
- Les registres suivants sont volatiles et peuvent donc être détruits dans la fonction appelée : RAX, RCX, RDX, RSI, RDI, R8 à R11, ST(0) à ST(7), XMM0 à XMM15.

Peut-on utiliser le coprocesseur mathématique (x87) et les MMX sous Windows ?

Auteurs : Neitsa ,

Contrairement à ce qu'avait annoncé dans un premier temps Microsoft la réponse finale et courte est : oui, sauf dans les drivers.

Pour préciser cette réponse, Microsoft annonçait que la commutation de contexte (context switch) effaçait les registres ST(x) et XMM. Toutefois cette information s'est révélée fautive et a été corrigée par un responsable du programme Visual C++ et un programmeur du groupe Kernel chez Microsoft :

```
"Let them know that the OS does preserve state of x87 and MMX registers on context switches."  
[...]
```

"For user threads the state of legacy floating point is preserved at context switch. But it is not true for kernel threads. Kernel mode drivers can not use legacy floating point instructions."

Ref. : <http://www.planetamd64.com/index.php?showtopic=3458&view=findpost&p=68756>

Comment est le stack frame avant et/ou après l'appel de fonction ?

Auteurs : Neitsa ,

La procédure (ou fonction) appelante est responsable de l'allocation de l'espace nécessaire sur la pile pour la procédure appelée. Elle (la procédure appelante) doit toujours allouer suffisamment d'espace pour les 4 registres en paramètres même si la procédure appelée ne nécessite pas autant d'arguments. La question est assez complexe; pour plus d'informations voir la référence ci-après :

Ref. : [http://msdn2.microsoft.com/en-us/library/ew5tede7\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/ew5tede7(VS.80).aspx)

Quel est le problème avec l'alignement de la pile en mode 64 bits avant d'appeler une API ?

Auteurs : Neitsa ,

Bien que la pile utilise des quadruples mots (8 octets soit 64 bits), le pointeur de pile (RSP) doit toujours être aligné sur un multiple de 16 avant d'appeler une API.

Ref. : AMD64 ABI Ref. : [http://msdn2.microsoft.com/en-us/library/ew5tede7\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/ew5tede7(VS.80).aspx)

Pour se prémunir de ce problème, on peut utiliser un code comme celui-ci :

```
PUSH RSP           ; sauvegarde la position courante de RSP sur la pile
PUSH [RSP]        ; garde une autre copie sur la pile
AND SPL,0F0h      ; ajuste RSP pour aligner la pile au cas où elle ne le serait pas.
                  ;
                  ; paramètre(s) de l'API s'il y a lieu.
                  ;
SUB RSP, 32d       ; ajuste RSP pour la sauvegarde des paramètres (32d = 0x20)
CALL API
ADD RSP, xx        ; nettoie la pile (avec xx = 32d + (nombre de paramètre poussés * 8))
POP RSP           ; restaure RSP à sa valeur originale.
```

N.B : Certaines APIs supportent une pile mal alignée sans déclencher d'exception. Certaines déclenchent une exception en interne et rétablissent (toujours en interne) une pile alignée. D'autres encore ne supportent aucunement une pile mal alignée. Dans le doute : alignez la pile avant l'appel.