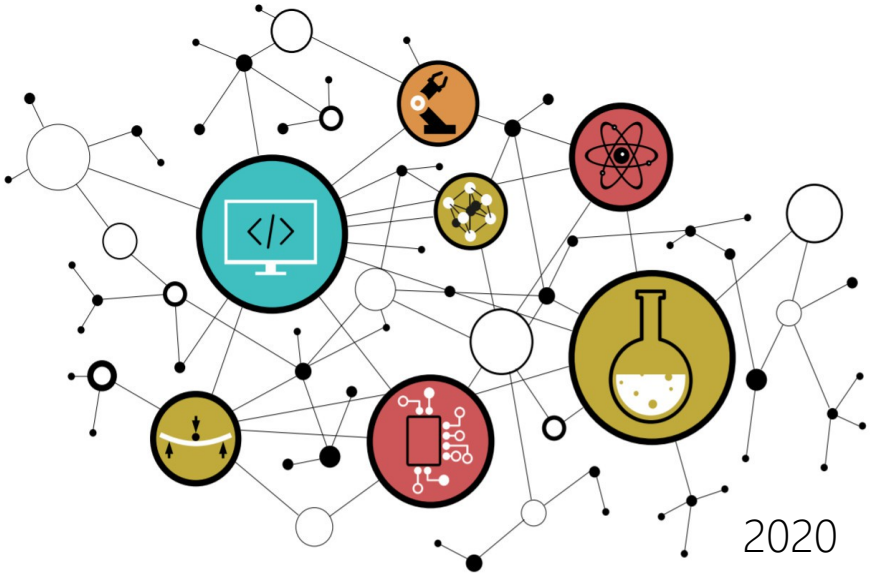
ENSI CAEN

ARCHITECTURES DES ORDINATEURS

MEMENTO



2020

MEMENTO

Syntaxe assembleur AT&T – GNU AS

La syntaxe AT&T est celle utilisée par GCC et donc celle de son assembleur AS (GAS ou GNU AS). Cette syntaxe est de nos jours standardisée sur tout système Unix-like. Unix a d'ailleurs été créé par l'entreprise AT&T (rachat en 2005) dans ses laboratoires Bell (filiale de AT&T). D'autres syntaxes assembleur permettant de représenter un même programme binaire x86/x64 existent. Prenons l'exemple de la syntaxe Intel, utilisée par exemple par ICC (Intel C++ Compiler) et également présente dans les documentations techniques du fondeur. Intéressons-nous à la syntaxe la plus standard de nos jours, la syntaxe AT&T, et prenons un exemple en assembleur x86/x64 :

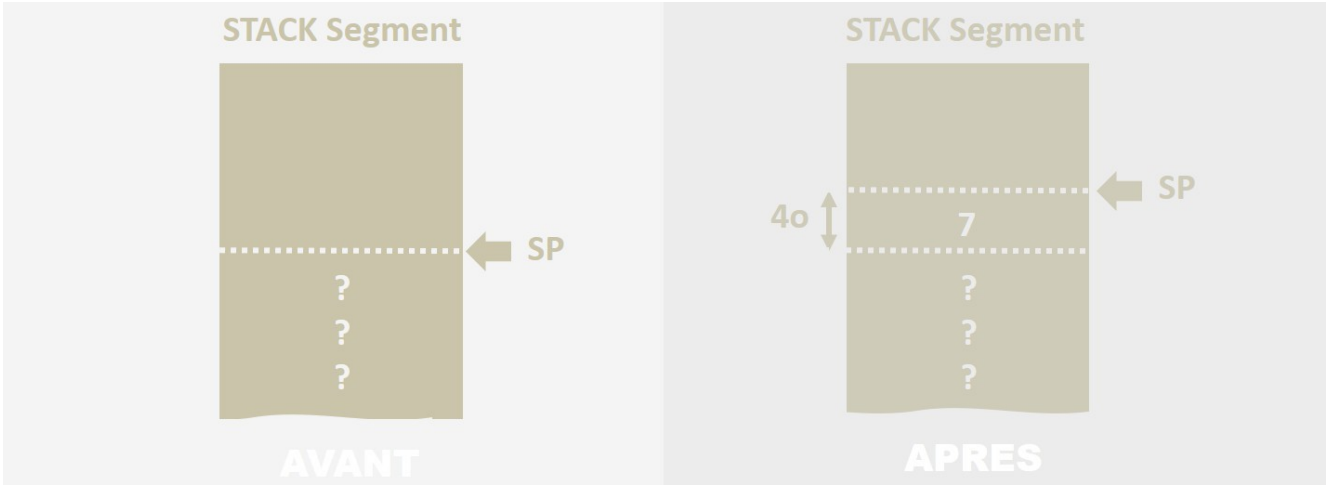
main : addl $0x07, %eax		
main:	addl	$0x07, %eax
Label :	mnemonic	Source, source/destination
Référence symbolique (simple chaîne de caractères) à l'adresse virtuelle en mémoire principale de l'instruction pointée. Il s'agit toujours de la première instruction suivant le label dans le programme assembleur	Nom de l'instruction assembleur à exécuter par le CPU	Opérandes sources et de destination. Une opérande sera toujours soit une constante (valeur ou pointeur), soit un registre (contenant une valeur ou un pointeur vers la mémoire principale)
Opération en pseudo-code	EAX = EAX + 7	

Observons les principaux préfixes et suffixes rencontrés dans la syntaxe AT&T. Durant une analyse ou un développement assembleur poussé sur un projet réel, une étude plus avancée de la syntaxe serait nécessaire. Seuls ses fondamentaux sont présentés ci-dessous :

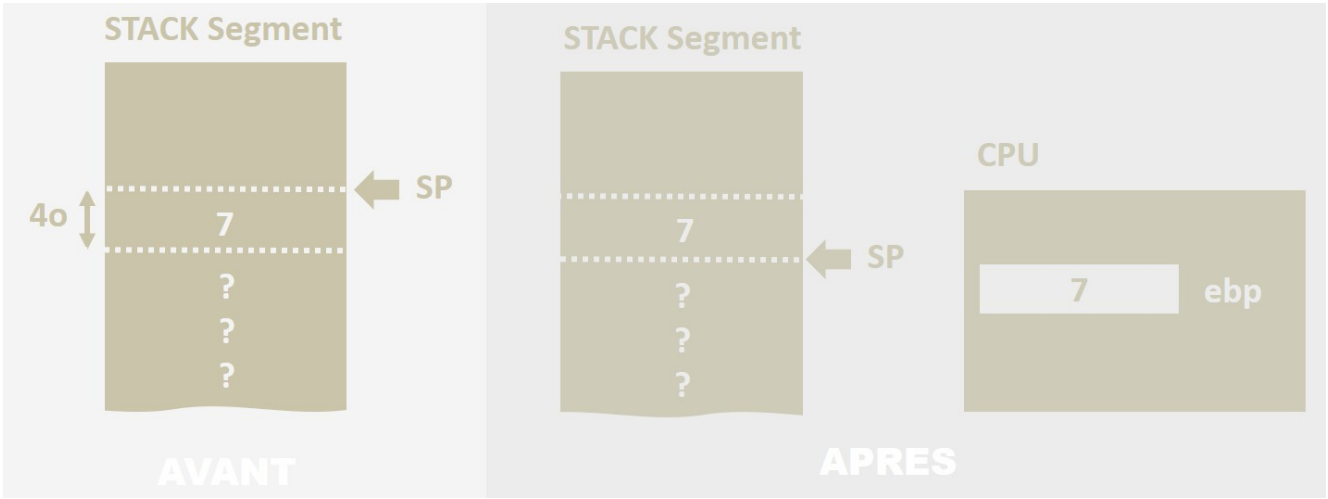
- **$** : préfixe spécifiant une constante (adressages immédiat/valeur et direct/pointeur)
- **%** : préfixe spécifiant un registre (adressages registre/valeur et indirect/pointeur)
- <*mnemonic*>**b** ou **s** ou **l** ou **q** : suffixe spécifiant la taille des opérandes manipulées (b=byte=8bits=1octet, s=short=16bits=2octets, l=long=32bits=4octets et q=quad=64bits=8octets)
- <*offset*>(<*register*>) : adressage indirecte vers la mémoire principal par registre. Il s'agira toujours d'un accès en mémoire principale (lecture ou écriture). Par exemple *-4(%ebp)* équivaut en pseudo-code à **(BP – 4)*, soit accès (lecture ou écriture) au contenu mémoire pointé par l'adresse contenue dans le registre EBP (pointeur BP), moins 4 octets.

Exemples d'instructions x86/x64

- *pushl $7* : écriture en mémoire principale sur le sommet de la pile.
- Opérations : déplacement du pointeur de sommet de pile (SP ou Stack Pointeur) du nombre d'octets spécifié par le suffixe de mnémonique (4 octets dans l'exemple donné), puis écriture en mémoire à la nouvelle adresse pointée par SP. Empile une donnée sur la pile courante.

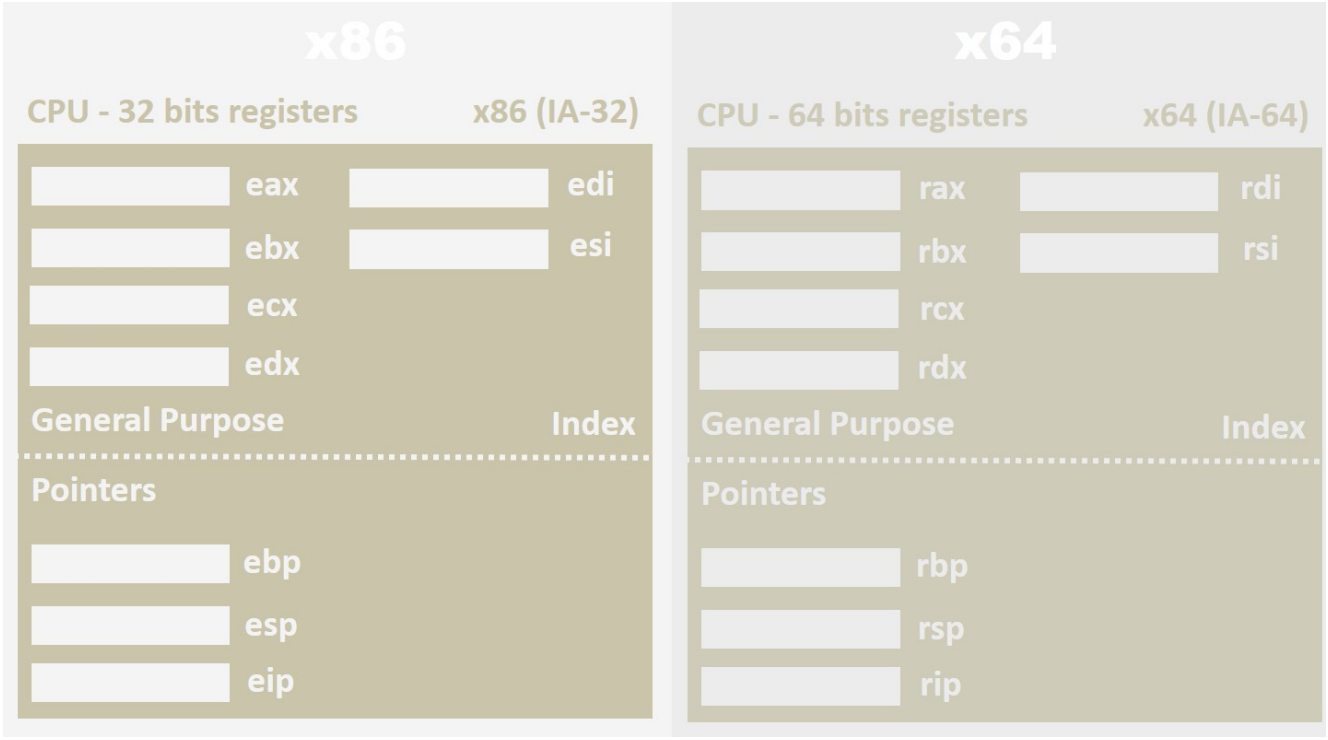


- *popl %ebp* : lecture en mémoire principale depuis le sommet de la pile.
- Opérations : Lecture mémoire à l'adresse pointée par SP du nombre d'octets spécifié par le suffixe du mnémonique (4 octets dans l'exemple donné), écriture vers le registre de destination présent dans le CPU, puis déplacement du pointeur de sommet de pile (SP ou Stack Pointeur) du nombre d'octets spécifié par le suffixe de mnémonique.



Environnement d'exécution (langage d'assemblage)

Observons une partie des registres présents dans un CPU 8-16-32bits/64bits sur architectures compatibles x86/x64. Les représentations ci-dessous sont non exhaustives et ne présentent par exemple pas les registres flottants (STx) et vectoriels (MMX, XMM, YMM, etc).



Pour des soucis de rétrocompatibilité, toute nouvelle architecture x64 doit rester compatible avec les générations antérieures x86. Cette rétrocompatibilité remonte jusqu'aux architectures 8-16bits, dont le 8086 fut un modèle clé (1978). Par exemple, les registres 16bits et 8 bits de ce processeur sont toujours supportés à notre époque. Prenons l'exemple du registre à usage général A (Accumulator), déjà présent sur Intel 4004 (1971), ainsi que ses déclinaisons 8, 16, 32 et 64 imbriquées les unes dans les autres (AL, AH, AX, EAX et RAX).