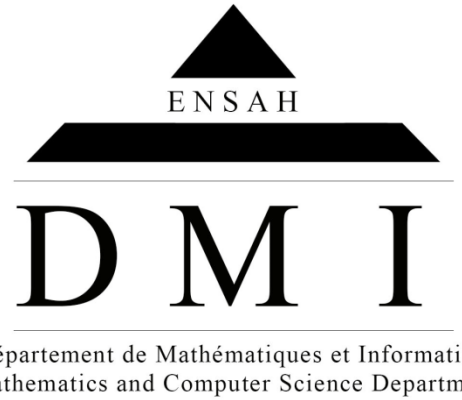
Architecture Intel
&
assembleur



Génie informatique E.N.S.A.H
Semestre 1
Année universitaire 2018/2019

Présentation

Les ordinateurs PC des début des années 80 étaient équipés du 8086, un processeur 16 bits.

Depuis, les processeur n'ont cesser d'évoluer, les modèles se succédaient. 80286, 80386, 80486, Pentium I, Pentium II ….Core I3,5,7,9....
Chaque processeur est plus puissant que les précédents; fréquence d'horloge plus élevée, bus de données plus large, nouvelles instructions, augmentation du nombre de registres …
Chacun de ces processeurs est compatible avec les modèles précédents.

Un programme écrit avec un langage machine d'un processeur peut s'exécuter sur les nouveaux modèles, mais l'inverse n'est pas possible.

Présentation

Langage machine et langage d'assemblage:
Le langage machine est une suite de bits interprétée par le processeur d'un ordinateur exécutant un programme informatique.

C'est le langage natif d'un processeur, c'est-à-dire le seul qu'il puisse traiter. Il est composé d'instructions et de données à traiter codées en binaire.

Notons que, chaque type de microprocesseur possède son propre jeu d'instructions

Convention Intel

Pour programmer en assembleur, nous sommes forcés de suivre certaines conventions, qui en réalité viennent directement du constructeur.

Le constructeur le plus populaire est ***Intel***

Comme exemple de programme, nous avons :

```
mov eax,101
mov ebx,0x378
mov ecx,3
mov edx,1
int 0x80
```

Le code généré est visiblement claire

Convention AT&T

Le style de codage AT&T a vu le jour en même temps qu'Unix et les conventions adoptées datent donc de la même époque.

Si nous avons pris l'habitude de coder en suivant les conventions d'Intel, nous aurions de la peine à se familiariser rapidement avec les conventions AT&T.

Comme exemple de programme, nous avons :

```
movl  %esp,%ebp
pushl %ebx
movl  %eax,%ebx
```

Pour les habitués du DOS, le code ci-dessus est « illisible »

Convention AT&T

Il existe des assembleurs grâce auxquels on peut utiliser les conventions Intel sur une plateforme Unis (linux).

Nasm (Netwide assembler) et l'un de ces assembleurs. Nous l'avons choisi pour les deux raisons suivantes:

- La convention Intel parait plus claire en comparaison avec la convention AT&T
- Nasm ne sera pas nostalgique pour ceux qui étaient habitués au DOS

Deux raisons pouvaient nous faire migrer vers la convention AT&T

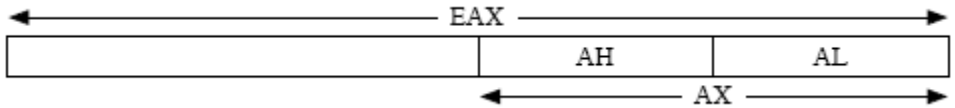
- Le format d'affichage GNU gdb.
- La syntaxe de l'assembleur inline à l'intérieur du code C via __asm__().

Les différents registre du microprocesseur

Le nombre de registres diffère d'un microprocesseur à un autre.
Les plus utilisés sont:

Les registres généraux

EAX : registre accumulateur (accumulator reg.) utilisé pour les opérations arithmétiques et le stockage de la valeur de retour des appels systèmes.
ECX : registre compteur (counter reg.)
EBX : registre de base (base reg.)
EDX : registre de données (data reg.) utilisé pour les opérations arithmétiques et les opérations d'E/S.
AX : 16 bits de poids faible de EAX (idem BX, CX, DX)
AL : octet de poids faible de AX (idem BL, CL, DL)
AH : octet de poids fort de AX (idem BH, CH ,DH)



Les différents registres du microprocesseur

D’autres registres sont utilisés comme pointeurs appelés:

Registres d’adresses

ESI : pointeur source (Extended Source Index)

EDI : pointeur destination (Extended Destination Index)

EBP : pointeur de base (Extended Base Pointer)

ESP : pointeur de pile (Extended Stack Pointeur)

Les différents registres du microprocesseur

Il existe d’autre registres qui sont utilisés pour d’autres fonctionnalitées:

EIP : pointeur d'instruction

EFLAGS : registre d‘états (drapeaux)

CS, SS, DS, ES, FS, GS : registres de segment (16 bits) : adresses et données de programme.

Les différents drapeaux du registre d’état

Les drapeaux du registre d’état sont les suivants:

Zero Flag (ZF)

1 si les deux opérandes utilisées sont égales, 0 sinon.

Overflow Flag (OF)

1 si le dernier résultat a provoqué un overflow, 0 sinon.

Carry Flag (CF)

: 1 si la dernière opération a généré une retenue ;0 sinon.

Sign Flag (SF)

:1 si la derniere opération a généré un resultat négatif, 0 s'il est positif ou nul.

Les différents drapeaux du registre d’état

Parity Flag (PF)

:1 si la derniere opération a généré un résultat impair, 0 s'il est pair (nombre de bits a 1).

Interrupt Flag (IF)

1 si les interruptions sont autorisées, à 0 sinon.

Position de quels que drapeaux dans le registre d'état

Drapeau	Nom	Position
cf	Carry Flag	0
pf	Parity Flag	2
af	Auxiliary carry Flag	4
zf	Zero Flag	6
sf	Sign Flag	8
if	Interruption Flag	9
df	Direction Flag	10
of	Overflow Flag	11

Le format des instructions

En assembleur, une instruction se compose d'un mnémonique qui désigne l'opération qui est suivi d'opérandes.

Instruction = opération + opérandes

Une opérande peut être :

Une donnée brute ou une adresse:

```
mov eax, 55   (décimal)
mov eax, 0b110111  (binaire)
mov eax, 0xfa89  (hexadécimal)
```

```
mov dx, 0x37A
```

Dans ce cas, 37A(en hexadécimal) désigne l'adresse d 'un registre du port parallèle.

Les modes d’adressage

Pour accéder aux données stockées en mémoire, on dispose de plusieurs mode d’adressage.

Adressage direct: dans ce cas, l’opérande est une adresse de 32 bits qui désigne le même emplacement en mémoire dont le contenu peut changer.

Exemple : mov eax, [0x0000f13a] : cette instruction a pour rôle de mettre le contenu de la mémoire d’adresse 0x0000f13a dans le registre eax.

Adressage par registre: dans ce cas l’opérande est un registre

Exemple : mov eax, esp : mettre dans eax l’adresse du sommet de la pile

Les modes d’adressage

Adressage indirect par registre: dans ce cas, l’opérande est un registre qui contient l’adresse d’une case mémoire:

mov eax, [esp] : cette instruction a pour rôle de mettre le contenu du sommet de la pile dans le registre eax.

Adressage indexé : dans ce cas l’opérande est une adresse mémoire contenu dans un registre associé à un décalage.
Mov eax,[ebp +8].

Les différentes instruction x86

Les instructions du x86 peuvent être regroupées en différentes catégories.

Les opérations de transfère : ce sont des opérations qui s’effectuent entre la mémoire et les registres

Les opérations arithmétiques

Les opérations logiques

Les opérations de décalage et de rotation.

Les opérations de branchement (saut conditionnel, saut inconditionnel, boucle et appels système.

Les opérations sur les chaînes de caractères.

Les instructions de transfère

Ce sont des instructions qui consistent à copier des données entre la mémoire et les registres. Le mnémonique utilisé est ***mov***

mov registre, mémoire

mov mémoire, registre

mov registre, registre

mov mémoire, mémoire est une instruction qui n'est pas permise

L'instruction xchg est utilisée pour échanger le contenu de deux registre ou le contenu d'un registre et d'une case mémoire.

xchg eax, ebx xchg eax, [0x65f28a66]

Les instructions de transfère

Les deux opérations push et pop concernent la pile

push est utilisée pour empiler

pop est utilisée pour dépiler

Remarque:

Plus on empile sur la pile, plus l’adresse du sommet de la pile décroit.

Les instructions arithmétiques

add est une instruction qui permet de réaliser une addition entière (en complément à 2).

Cette instruction nécessite deux opérandes dont au moins un registre.

Cette opération positionne les drapeaux CF et OF

add eax,0x25fe

Attention aux incompatibilités:
add al, ebx registres de différentes tailles.

Les instructions arithmétiques

mul est une instruction qui permet de réaliser une multiplication entière positive

Elle nécessite une seule opérande. Elle réalise le produit du contenu du registre eax avec l'opérande en question.

mul ebx

Le résultat est stocké dans les deux registres edx|eax

mul ebx edx|eax ebx.eax

Les instructions arithmétiques

Pour réaliser une multiplication entière en complément à 2 on dispose de l'instruction imul.

Elle possède les mêmes caractéristiques que mul, sauf qu'on utilise des entiers relatifs.
Elle nécessite une seule opérande. Elle réalise le produit du contenu du registre eax avec l'opérande en question.

imul ebx

Le résultat est stocké dans les deux registres edx|eax

imul ebx edx|eax ebx.eax

Les instructions arithmétiques

`add`	*dst*	*src*	ajoute *src* à *dst*
`adc`	*dst*	*src*	ajoute *src* à *dst* avec retenue
`sub`	*dst*	*src*	soustrait *src* à *dst*
`sbb`	*dst*	*src*	soustrait *src* à *dst* avec retenue
`mul`	*src*		multiplie `eax` par *src* (résultat dans `edx`|`eax`)
`imul`	*src*		multiplie `eax` par *src* (cplt à 2)
`div`	*src*		divise `edx`|`eax` par *src* (`eax`=quotient, `edx`=reste)
`idiv`	*src*		divise `edx`|`eax` par *src* (cplt à 2)
`inc`	*dst*		$1 + dst$
`dec`	*dst*		$dst - 1$
`neg`	*dst*		$-dst$

Les instructions logiques

Les opérations logiques sont des opérations bit à bit.

Opération destination, source

L'opération and possède deux opérandes (destination et source).

Elle peut être utilisée comme masque pour extraire un ensemble de bits.

Comme exemple and eax, 0b11110000 qui permet d'extraire les quatre bits du registre eax à partir du cinquième.

Nous avons aussi, or, xor et not.

Les instructions logiques

`not`	dst		place (not dst) dans dst
`and`	dst	src	place (src AND dst) dans dst
`or`	dst	src	place (src OR dst) dans dst
`xor`	dst	src	place (src XOR dst) dans dst

Les instructions de décalage et de rotation

Ces instructions nécessitent deux opérandes; un registre et le nombre de bits de décalage.

Décalage logique à gauche shl: cette opération insère nb 0 à partir de la droite.

Exemple shl ah, 3 (ah=01100111 ah=00111000)

Décalage arithmétique à droite sar : cette opération insère nb copies du bit qui se trouve complètement à gauche (bit du poids fort).

Exemple sar al, 5 (al=10100001 al=11111101)

Les instructions de décalage et de rotation

Rotation à gauche rol : cette opération fait une rotation de nb bits, les bits qui sortent de la gauche sont réinjectés à partir de la droite.

Exemple rol bl, 2 (bl=10111001 □ bl=11100110).

Rotation à droite avec retenue crr: cette opération permet de faire une rotation de nb bits à droite en prenant en considération le bit de la retenue.

Exemple crr bl, 4 : (bl=01101110 c=1 □ bl=11010110 c=1)

1er décalage : (bl=01101110 c=1 □ bl=10110111 c=0)

2ème décalage : (bl=10110111 c=0 □ bl=01011011 c=1)

3ème décalage : (bl=01011011 c=1 □ bl=10101101 c=1)

4ème décalage : (bl=10101101 c=1 □ bl=11010110 c=1)

Les instructions de décalage et de rotation

`sal`	*dst*	*nb*	décalage arithmétique à gauche de *nb* bits de *dst*
`sar`	*dst*	*nb*	décalage arithmétique à droite de *nb* bits de *dst*
`shl`	*dst*	*nb*	décalage logique à gauche de *nb* bits de *dst*
`shr`	*dst*	*nb*	décalage logique à droite de *nb* bits de *dst*
`rol`	*dst*	*nb*	rotation à gauche de *nb* bits de *dst*
`ror`	*dst*	*nb*	rotation à droite de *nb* bits de *dst*
`rcl`	*dst*	*nb*	rotation à gauche de *nb* bits de *dst* avec retenue
`rcr`	*dst*	*nb*	rotation à droite de *nb* bits de *dst* avec retenue

Les instructions de comparaison

L'instruction de comparaison cmp compare deux opérandes en faisant une soustraction des deux opérandes sans stocker le résultat. Elle positionne le drapeau CF.

Exemple cmp eax, ebx : l'opération réalisée est eax- ebx

Si eax=ebx ZF=0 sinon ZF=1

Instructions de branchement

Il existe plusieurs instructions de branchement.

- Les sauts conditionnels.
- Les sauts inconditionnels.
- Les boucles fixes.
- Les boucles conditionnelles.
- Les boucles inconditionnelles.

Le saut conditionnel réalise un saut vers l'étiquette spécifiée lorsque la condition est remplie.

L'expression de l'instruction est comme suit: jxxxx etiquette.

Exemple
je étiquette : saut à étiquette si le drapeau d'égalité est à 1

jne étiquette : saut à l'étiquette si le drapeau d'égalité est à 0

Pour ces deux instructions, le drapeau d'égalité doit être positionné par l'instruction cmp.

jge étiquette : saut à étiquette si c'est supérieur ou égal.

jnge étiquette : saut à l'étiquette si c'est inférieur.

Avant d'utiliser ces deux instructions, il faut utiliser l'instruction cmp.

jl étiquette: saut à étiquette si c'est inférieur strictement.

jnl étiquette: saut à étiquette si ce n'est pas strictement inférieur.

jo étiquette: saut à étiquette si overflow (OF=1).

jno étiquette : saut à étiquette si no overflow (OF=0).

jc étiquette: saut à étiquette si le drapeau Carry est positionné (CF=1).

jnc étiquette: saut à étiquette si le drapeau Carry est non positionné (CF=0).

jp étiquette : saut à étiquette si le drapeau parity est positionné (PF=1).

jnp étiquette :saut à étiquette si le drapeau parity n'est pas positionné (PF=0).

jcxz étiquette : saut à étiquette si le registre cx est nul.

jecx étiquette : saut à étiquette si le registre ecx est nul.

L’instruction loop est une instruction qui exécute une boucle fixe. loop étiquette : elle décrémente le contenu du registre ecx est saut à étiquette si ecx ≠ 0.

L’instruction loope est une instruction qui exécute une boucle conditionnelle.

loope étiquette : décrémente le registre ecx (ecx ecx-1) est saute à étiquette si (ecx ≠0 et ZF=1).

loopne étiquette : décrémente le registre ecx (ecx ecx-1) est saute si (ecx ≠0 et ZF=0).

`cmp`	*sr*1	*sr*2	compare *sr*1 et *sr*2
`jmp`	*adr*		saut vers l'adresse *adr*
`jxx`	*adr*		saut conditionné par xx vers l'adresse *adr*

`loop`	*adr*	répétition de la boucle *nb* de fois (*nb* dans `ecx`)
`loopx`	*adr*	répétition de la boucle conditionnée par x