ODO structure ordi

Charline DANTINNE

23 mars 2021

1 La structure d'un ordinateur

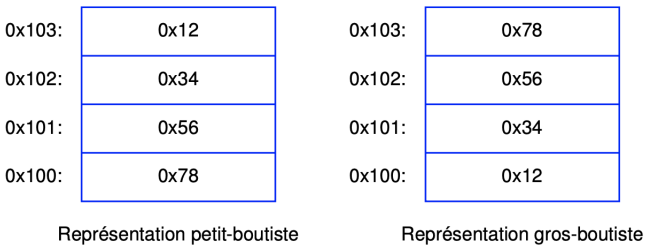
Architecture x86-64
→ chaque cellule contient 8 bits
→ l'adressage s'effectue sur 64 bits.
L'espace d'adressage correspond donc à l'intervalle :

$$[0, 2^6 4 - 1]$$

Le stockage de données sur plus d'une cellule

1. Représentation petit-boutiste → depuis le poids faible vers le poids fort
2. Représentation gros-boutiste → depuis le poids fort vers le poids faible

Exemple: Représentation de 0x12345678 sur des cellules de 8 bits, à partir de l'adresse 0x100:



Même si la mémoire est organisée en octet, les échanges entre le porcesseur et la mémoire externe s'effectuent par blocs de plus grande taille.

Une donnée représentée sur n octets, où n est une puissance de 2, est dite **alignée** si son adresse est un multiple de n.

Certaines architectures interdisent les transferts de données non alignées. D'autres (ex : x86-64) le permettent mais les transferts ne sont pas efficaces.

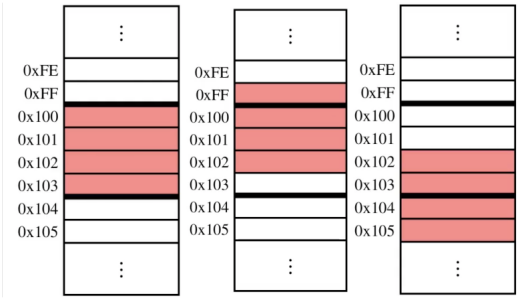
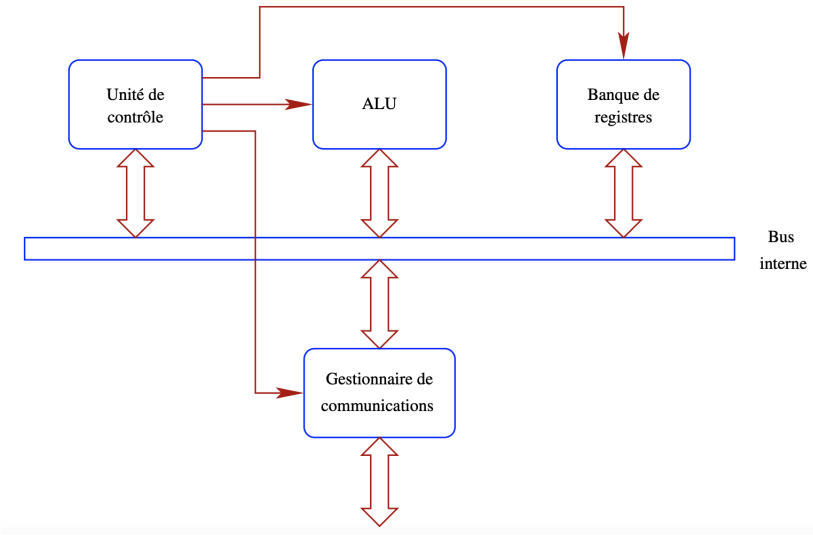


FIGURE 1 – Défaut d'alignement (les 2 fig de Gauche)

1.1 Structure d'un processeur



1. **La banque de registre :** petite quantité de mémoire vive, utilisée comme espace de travail. L'ensemble des registres disponibles dépend de l'architecture du processeur.
2. **Unité arithmétique et logique** (Arithmetic Logic Unit, ALU) est le composant chargé de traiter l'information.
3. **Le bus interne** est un canal de communication entre les composants du porcesseur. Sa taille est la quantité de données qu'il est capable de communiquer en 1 seule opération élémentaire
4. **Le gestionnaire de communication** relie le bus interne à l'interface extérieure du processeur. Il est responsable de gérer les échanges de données avec la mémoire et les périphériques.
5. **L'unité de contrôle** est responsable de l'exécution des instructions en commandant les autres composants.

1.2 Les registres de contrôle

Pour exécuter les instructions, l'unité de contrôle du processeur gère 2 registres :

- Le **registre d'instruction** (*Instruction Register, IR*) contient le code de l'instruction (opcode) en cours d'exécution
- Le **compteur de programme** (*Programm Counter, PC, ou Instruction Pointer, RIP pour x86-64*) contient l'adresse en mémoire de la prochaine instruction à exécuter.

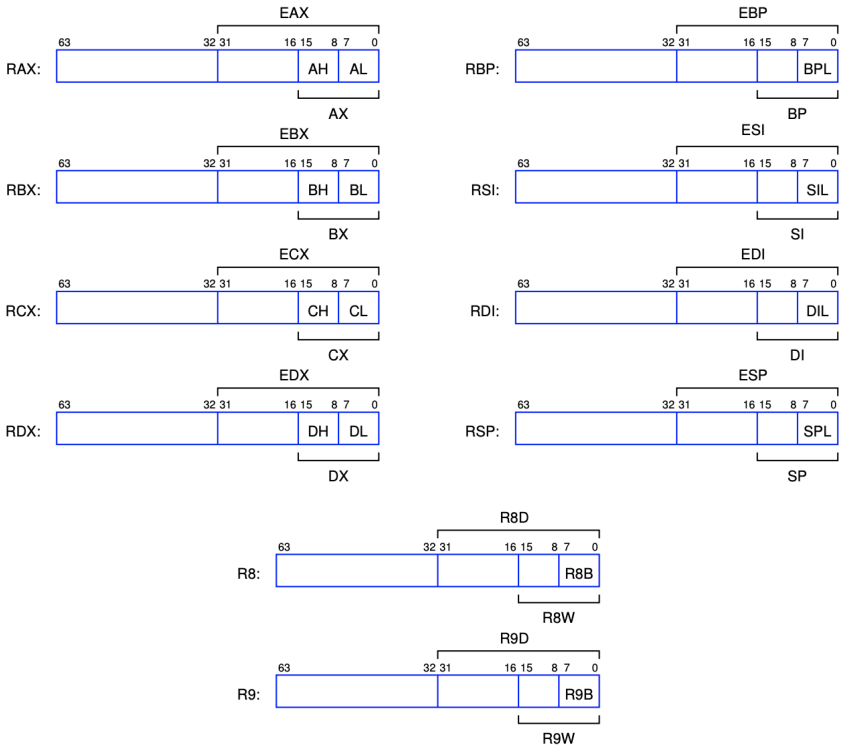
1.3 L'exécution des instructions

L'unité de contrôle effectue les opérations suivantes au rythme du signal d'horloge :

1. **Charger** dans IR l'opcode pointé par le PC
2. **Décoder** la valeur de IR
3. **Exécuter** l'instruction correspondante contrôlant les autres composants du processeur
   Si l'instruction possède des paramètres, leur lecture et leur décodage font partie de cette étape.
   Mettre PC à jour de façon à le faire pointer vers l'instruction suivante.

1.4 L'architecture x86-64

1.4.1 Les parties de registres



1.4.2 Les drapeaux

Le registre RFLAGS (registre de 64 bits) contient des drapeaux (flags) qui sont des bits d'info mis à jour par certaines instructions :

- **CF** (*Carry Flag*) : indique qu'une opération arithmétique sur des nombres de n bits produits un **report à la position n**.
- **ZF** (*Zero Flag*) : indique qu'une opération a fourni un **résultat nul**
- **SF**(*Sign Flag*) : correspond au bit de poids fort du résultat d'une opération (**bit de signe** dans le cas d'une **données signées**)
- **OF**(*Overflow Flag*) : indique un **dépassement arithmétique** pour des données signées

1.4.3 Les modes d'adressage

L'adressage registre Une opérande est lue depuis un **registre** (source), ou qu'un résultat doit être placé dans un registre (destination).

L'adressage immédiat Définit une opérande constante

L'adressage direct Une opérande doit être lue depuis la **mémoire** (source), ou qu'un résultat doit être écrit en mémoire (destination), à une adresse fixée.

Syntaxe : <*taille*> ptr [<*adresse*>]

<*adresse*> est un **pointeur** vers l'emplacement de mémoire concerné
<*taille*> est un des mots clés :
— qword → 64 bits
— dword → 32 bits
— word → 16 bits
— byte → 8 bits
△ les opérandes source et destination doivent être de même taille

△ l'architecture x86-64 est petit-boutiste
△ Il ne faut pas oublier d'aligner si nécessaire les données mémorisées sur plus d'une cellule.

L'adressage indirect Indique un accès à la mémoire. Une opérande doit être lue depuis la **mémoire** (source), ou qu'un résultat doit être écrit en mémoire (destination), à une adresse donnée par le contenu d'un **registre**.

△ L'adressage indirect ne peut faire intervenir que des registres de 64 bits.

L'adressage indirect indexé Variante de l'adressage indirect. L'expression du pointeur peut faire intervenir un **index**, un **facteur** et un **déplacement**
Syntaxe :
<*taille*> ptr [<*base*> + <*facteur*> * <*index*> ± <*déplacement*>]
où
- <*taille*> est qword, dword, word, byte
- <*base*> et <*index*> sont des **registres de 64 bits**
- <*facteur*> vaut 1, 2, 4 ou 8
- <*déplacement*> est une **constante** signée représentable sur 32 bits

1.4.4 Les instructions x86-64

Pour chaque instruction, nous préciserons,
- les **modes d'adressage** qu'elle supporte, à l'aide des codes suivants :
  — *imm* pour immédiat
  — *reg* pour registre
  — *mem* pour direct, indirect ou indirect indexé
- les **drapeaux** affectés par son exécution

Les instructions de manipulation de données

MOV

Cette instruction **déplace** des données, de la seconde opérande vers la première.

Mode d'adressage

Op.1	Op.2
reg	*imm*
mem	*imm*
reg	*reg*
reg	*mem*
mem	*reg*

△ Il n'est pas permis de combiner des accès à la mémoire pour les 2 opérandes (mem|mem)

Drapeaux affectés : Aucun

XCHG

Cette instruction **échange** le contenue de ses 2 opérandes.

XCHG EAX, EAX n'a aucun effet. Elle s'abrévie en NOP (No OPeration)

Mode d'adressage

Op.1	Op.2
reg	*reg*
reg	*mem*
mem	*reg*

△ Pas de possibilité d'adressage immédiat.

Drapeaux affectés : Aucun

ADD

Cette instruction **ajoute** sa seconde opérande à la première.

La même instruction peut être employée pour des nombres non signés ou bien représentés par complément à deux.

Mode d'adressage

Op.1	Op.2
reg	*imm*
mem	*imm*
reg	*reg*
reg	*mem*
mem	*reg*

Drapeaux affectés : ZF, CF, SF et OF

SUB

Cette instruction **soustrait** sa seconde opérande à la sa première.

Mode d'adressage

Op.1	Op.2
reg	*imm*
mem	*imm*
reg	*reg*
reg	*mem*
mem	*reg*

Drapeaux affectés : ZF, CF, SF et OF

CMP

Cette instruction **soustrait** sa seconde opérande à la première mais sans la modifier. Le seul effet de cette opération est de mettre à jour les drapeaux.

Cela permet d'effectuer des **comparaisons** de valeurs, pouvant servir de base de décision dans les instructions suivantes.

Mode d'adressage

Op.1	Op.2
reg	*imm*
mem	*imm*
reg	*reg*
reg	*mem*
mem	*reg*

Exemple:

L'instruction

```
CMP EAX, EBX
```

calcule la différence Δ = EAX – EBX. On a alors:

- ZF = 1 ssi $\Delta = 0$, c'est-à-dire EAX = EBX.
- SF = 1 ssi $\Delta < 0$, c'est-à-dire EAX < EBX (si les nombres sont signés).

Drapeaux affectés : ZF, CF, SF et OF

INC et DEC

Ces instructions **incrémentent** ou **décrémentent** leur opérande, qui sert donc de source et de destination.

Mode d'adressage

Op.1
reg
mem

Drapeaux affectés : CF n'est pas affecté et ZF, SF et OF sont mis à jour

MUL

Cette instruction **multiplie** 2 nombres non signés de n bits, avec $n \in \{8, 16, 32, 64\}$. Le résultat est sur 2n bits.

Mode d'adressage

Op.1
reg
mem

L'opération effectuée dépend de la taille de cette opérande :

- 8 bits : Opérande multipliée par AL ; résultat placé dans AX
- 16 bits : Opérande multipliée par AX ; résultat placé dans DX :AX
- 32 bits : Opérande multipliée par EAX ; résultat placé dans EDX :EAX
- 64 bits : Opérande multipliée par RAX ; résultat placé dans RDX :RAX

△ Obligatoire et prédéfini

Contrairement à l'addition, le fait que les nombres sont représentés de façon signée ou non signée influence la multiplication

Drapeaux affectés :

CF et OF sont mis à 0 sir le résultat de l'opération est **représentable sur n bits**, où n est la taille des opérandes, et à 1 sinon.

ZF et SF sont modifiés de façon arbitraire.

IMUL

Cette instruction est similaire à MUL, mais calcule le produits de 2 nombres **signés**

L'opération effectuée, les modes d'adressages et les drapeaux sont identiques à ceux de MUL

Les instructions logiques

AND, OR et XOR

Ces instructions appliquent une **opération booléenne** bit par bit à leurs 2 opérandes, et écrivent le résultat dans la première.

- AND : Le résultat est égal à 1 ssi les **2 opérandes** sont égales à 1 (et logique)
  → forcer à 0 les bits
- OR : Le résultat est égal à 1 ssi **au moins une opérande** est égale à 1 (ou inclusif)
  → forcer à 1 les bits
- XOR : Le résultat est égal à 1 ssi **exactement une opérande** est égale à 1 (ou exclusif)
  → inverser les bits

x	y	AND(x, y)
0	0	0
0	1	0
1	0	0
1	1	1

x	y	OR(x, y)
0	0	0
0	1	1
1	0	1
1	1	1

x	y	XOR(x, y)
0	0	0
0	1	1
1	0	1
1	1	0

Mode d'adressage

Op.1	Op.2
reg	*imm*
mem	*imm*
reg	*reg*
reg	*mem*
mem	*reg*

Drapeaux affectés :
- CF et OF sont mis à 0
- ZF et SF sont mis à jour en fonction du résultat de l'opération

NOT

Son effet est d'**inverser** tous les bits de cette opérande (càd de le remplacer par son complément à 1)

x	NOT(x)
0	1
1	0

Mode d'adressage

Op.1
reg
mem

Drapeaux affectés : Aucun

Les instructions de manipulation de la pile

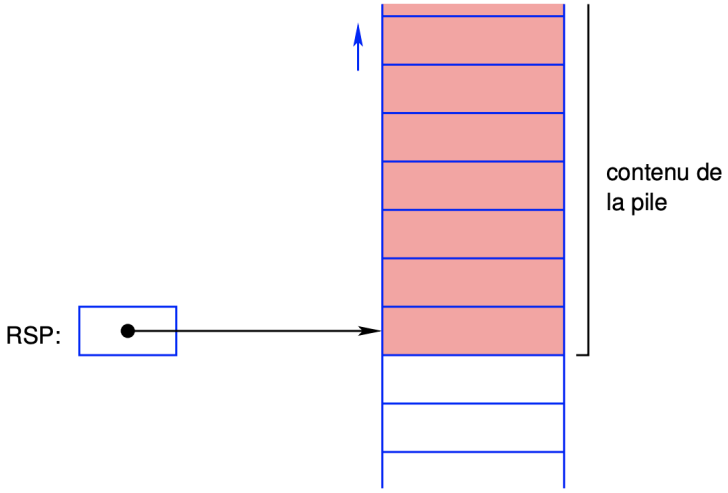
Les processeurs x86-64 gèrent une pile.
Une pile est une structure de données LIFO (Last In First Out), définissant 2 opérations :

- **Empiler** une valeur (push) à son sommet
- **Dépiler** une valeur (pop) depuis son sommet

La pile sert notamment à :

- Mémoriser des **données temporaires** comme les paramètres, les variables et les points de retour des fonctions invoquées par un programme
- **Sauvegarder** le contenu des registres modifiés par une sous-routine

- Le contenu d'une pile correspond a des **cellules consécutives**
- La pile croît vers les **adresses décroissantes**
- Le registre RSP pointe en permanence vers le **dernier octet** empilé



PUSH

Cette instruction **empile** une valeur de 64 bits, donnée par son opérande.

Mode d'adressage :

Op.1
imm
reg
mem

L'adressage registre doit utiliser un registre de 64 bits. Les adressages direct, indirect et indirect indexé doivent employer le mot-clé qword.

Opération réalisées :

1. **Décrémenter** RSP de 8 unités
2. **Recopier** l'opérande à l'endroit pointé par RSP

Drapeaux affectés : Aucun

POP

Cette instrcution **dépile** une valeur de 64 bits et l'écrit à l'endroit spécifié par son opérande.

Mode d'adressage :

Op.1
reg
mem

Opérations réalisées :

1. **Lire 8 octets** depuis l'emplacement pointé par RSP, et les recopier à l'endroit représenté par l'opérande
2. **Incrémenter** RSP de 8 unités

Drapeaux affectés : Aucun

Les instructions de contrôle

JMP

Cette instruction effectue un **saut inconditionnel** vers un emplacement de la mémoire de programme, donné par son opérande.

En d'autres termes, l'instruction charge cette opérande dans le **compteur de programme**

Mode d'adressage :

Op.1
imm
reg
mem

Drapeaux affectés : Aucun

En pratique, la destination d'un saut est exprimée symboliquement à l'aide d'une étiquette

Les instructions de saut conditionnel

Cette instruction effectue un **saut** vers un emplacement de la mémoire de programme que si une **condition particulière** est satisfaite

La condition peut porter sur **l'état d'un drapeau** :

Instruction	Condition
`JC`	CF = 1
`JNC`	CF = 0
`JZ`	ZF = 1
`JNZ`	ZF = 0
`JS`	SF = 1
`JNS`	SF = 0
`JO`	OF = 1
`JNO`	OF = 0

La condition peut aussi porter sur le résultat d'une **comparaison** réalisée par CMP :

Instruction	Condition	
`JE`	$op1 = op2$	
`JNE`	$op1 \neq op2$	
`JG`	$op1 > op2$	(valeurs signées)
`JGE`	$op1 \geq op2$	(valeurs signées)
`JL`	$op1 < op2$	(valeurs signées)
`JLE`	$op1 \leq op2$	(valeurs signées)
`JA`	$op1 > op2$	(valeurs non signées)
`JAE`	$op1 \geq op2$	(valeurs non signées)
`JB`	$op1 < op2$	(valeurs non signées)
`JBE`	$op1 \leq op2$	(valeurs non signées)

Drapeaux affectés : Aucun

LOOP

Cette instruction permet d'implémenter une **boucle** Son opérande est définie comme celle de JMP

Mode d'adressage :

Op.1
imm
reg
mem

Opérations réalisées :

1. **Décrémenter** RCX d'1 unité
2. Si **nouvelle valeur de RCX** est non nulle, effectuer un saut à l'endroit spécifié par l'opérande

— Le registre employé comme **compteur de boucle** est nécessairement RCX
— La valeur de RCX au départ ne correspond pas toujours au nombres d'itérations. En effet, pour RCX = 0, ce nombre est égal à 2^64

Drapeaux affectés : Aucun

CALL et RET

Ces instructions permettent de programmer des **sous-routines**

L'instruction CALL, mêmes modalités d'utilisation et modes d'adressage que JMP

Opérations réalisées :

1. Empiler la valeur courante de RIP, càd l'adresse où **reprendre l'exécution du programme** après la sous-routine
2. Effectuer un **saut** vers l'endroit donné par son opérande

L'instruction RET ne prend pas d'argument

Opérations réalisées :

1. **Dépiler** une valeur de 64 bits
2. Effectuer un **saut** vers cette adresse

Drapeaux affectés : Aucun