

Chapitre 0

La representation des nombres.

Les bases

Binary	Decimal	Hexadecimal	Binary	Decimal	Hexadecimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

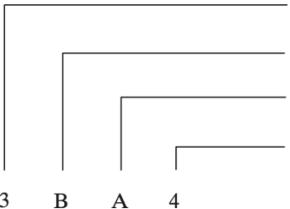
Convertir un entier en binaire (base 2)

Division	Quotient	Reste	2 ⁿ
37 / 2	18	1	2 ⁵
18 / 2	9	0	2 ⁴
9 / 2	4	1	2 ³
4 / 2	2	0	2 ²
2 / 2	1	0	2 ¹
1 / 2	0	1	2 ⁰

37 =	1	0	0	1	0	1
	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
37 =	32			+4		+1

On réécrit le binaire en partant du bas

Convertir un hexadécimal (base 16) en entier base 10

	$3 \times 16^3 = 12,288$
	$11 \times 16^2 = 2,816$
	$10 \times 16^1 = 160$
	$4 \times 16^0 = + 4$
	<hr/>
	Total: 15,268

Division	Quotient	Remainder
422 / 16	26	6
26 / 16	1	A
1 / 16	0	1

$1A6 = 1 \times 16^2 + 10 \times 16^1 + 6 \times 16^0$
 $= 256 + 160 + 6 = 422$

Ce nombre binaire **0001 0110 1010 0111 1001 0100**, en entier **1 484 692**, s'écrit plus facilement en hexadécimal **16A794**

1	6	A	7	9	4
0001	0110	1010	0111	1001	0100

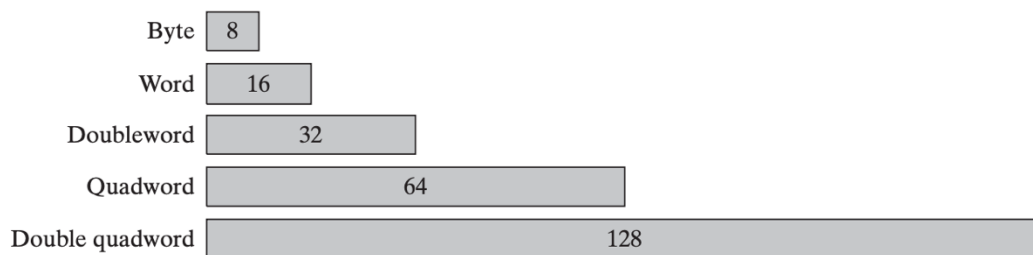
Additions

Retenue		1	1			
A	12	0	1	1	0	0
B	14	0	1	1	1	0
A+B	26	1	1	0	1	0

Retenue	1		
A	6	A	2
B	4	9	A
A+B	B	3	C

$A + 9 = 10 + 9 = 19 = 1 \times 16 + 3$. D'où la retenue 1

La taille de stockage des entiers



Les nombres binaires négatifs

Le premier bit représente le signe (1 = négatif, 0 = positif)

Négatif

1	1	1	0	1	1	1	0
---	---	---	---	---	---	---	---

Positif

0	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Le complément à 2

Un nombre binaire	00000001
Etape 1 : on inverse les bits	11111110
Etape 2 on ajoute 1	11111110 +00000001
On obtient son complément à 2	11111111

Pareille pour le complément à 16

6A3D --> 95C2 + 1 --> 95C3

95C3 --> 6A3C + 1 --> 6A3D

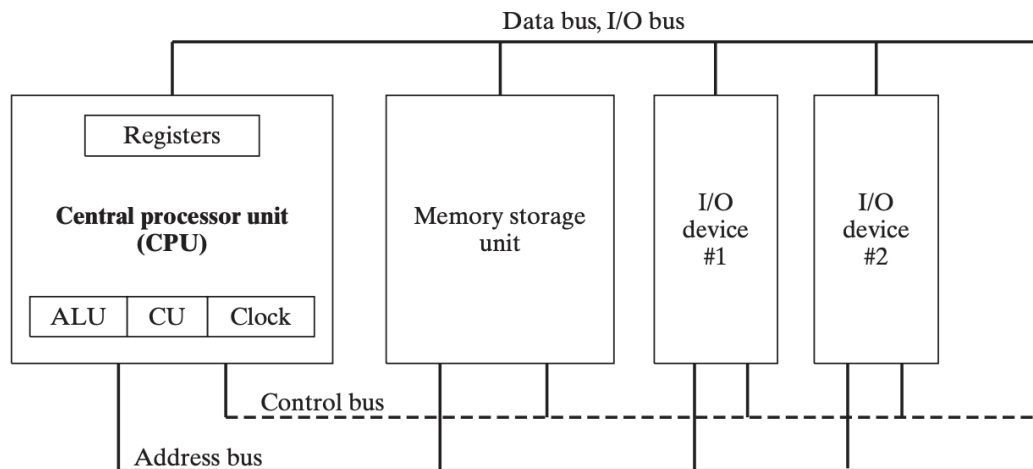
Pour retrouver le nombre entier négatif correspondant à une chaîne binaire commençant par 1 on cherche son complément à 2

La valeur binaire	11110000
Étape 1 : on inverse les bits	00001111
Étape 2: on ajoute 1	00001111 +1
Étape 3 : on obtient le complément	00010000
Étape 4 : on converti en entier	16

Donc 11110000 correspond à -16

Chapitre 2 Architecture d'un processeur

Figure 1



Ce chapitre décrit l'architecture de la famille de processeurs x86 et de son système informatique hôte du point de vue du programmeur.

Ce groupe comprend tous les processeurs Intel IA-32 et Intel 64, tels que les processeurs Intel Pentium et Core-Duo, ainsi que les processeurs Advanced Micro Devices (AMD), tels que Athlon, Phenom, Opteron et AMD64.

Le langage d'assemblage est un excellent outil pour apprendre comment fonctionne un ordinateur, et il vous oblige à avoir une connaissance pratique du matériel informatique.

À cette fin, les concepts et les détails de ce chapitre vous aideront à comprendre le code du langage assembleur que vous écrivez. Nous cherchons à trouver un équilibre entre les concepts applicables à tous les systèmes micro-informatiques et les spécificités des processeurs x86.

Vous pouvez travailler sur divers processeurs à l'avenir, nous vous exposons donc à des concepts généraux. Pour éviter de vous donner une compréhension superficielle de l'architecture des machines, nous nous concentrons sur les spécificités du x86, ce qui vous donnera une base solide lors de la programmation en langage assembleur.

La figure 1 montre la conception de base d'un micro-ordinateur hypothétique.

L'unité centrale de traitement (CPU), où les calculs et les opérations logiques ont lieu, contient un nombre limité d'emplacements de stockage nommés registres, une horloge haute fréquence, une unité de commande et une unité arithmétique et logique.

- L'horloge synchronise les opérations internes de la CPU avec les autres composants du système.

- L'unité de contrôle (CU) coordonne le séquençement des étapes impliquées dans l'exécution des instructions de la machine
- L'unité arithmétique et logique (ALU) effectue des opérations arithmétiques telles que l'addition et la soustraction et les opérations logiques telles que ET, OU et NON.

Le processeur est connecté au reste de l'ordinateur via des broches attachées au socket du processeur de la carte mère de l'ordinateur.

La plupart des broches se connectent au bus de données, au bus de contrôle et au bus d'adresses.

L'unité de stockage de mémoire est l'endroit où les instructions et les données sont conservées pendant l'exécution d'un programme informatique.

L'unité de stockage reçoit des demandes de données de la CPU, transfère les données de la mémoire vive (RAM) vers la CPU et transfère les données de la CPU dans la mémoire.

Tout le traitement des données a lieu dans la CPU, de sorte que les programmes résidant en mémoire doivent être copiés dans la CPU avant de pouvoir s'exécuter.

Des instructions de programme individuelles peuvent être copiées dans la CPU une par une ou des groupes d'instructions peuvent être copiés ensemble.

Un bus est un groupe de fils parallèles qui transfèrent des données d'une partie de l'ordinateur à une autre. Un système informatique contient généralement quatre types de bus : données, E / S, Contrôle et adresses.

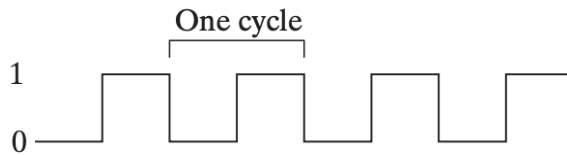
- Le bus de données transfère les instructions et les données entre la CPU et la mémoire.
- Le bus d'E / S transfère les données entre la CPU et les périphériques d'entrée / sortie du système.
- Le bus de contrôle utilise des signaux binaires pour synchroniser les actions de tous les appareils connectés au bus système.
- Le bus d'adresses contient les adresses des instructions et des données lorsque l'instruction en cours d'exécution transfère des données entre la CPU et la mémoire.

L'horloge

Chaque opération impliquant le CPU et le bus système est synchronisée par une horloge interne pulsant à une vitesse constante. L'unité de temps de base pour les

instructions machine est un cycle machine (ou cycle d'horloge). La longueur d'un cycle d'horloge est le temps nécessaire pour une impulsion d'horloge complète.

Dans la figure suivante, un cycle d'horloge est représenté comme le temps entre un front descendant et le suivant :



Le cycle d'exécution d'une instruction

Une seule instruction machine ne s'exécute pas comme par magie en une seule fois. La CPU doit passer par une séquence prédéfinie d'étapes pour exécuter une instruction machine, appelée cycle d'exécution des instructions. Supposons que le registre du pointeur d'instruction contienne l'adresse de l'instruction que nous voulons exécuter. Voici les étapes pour l'exécuter :

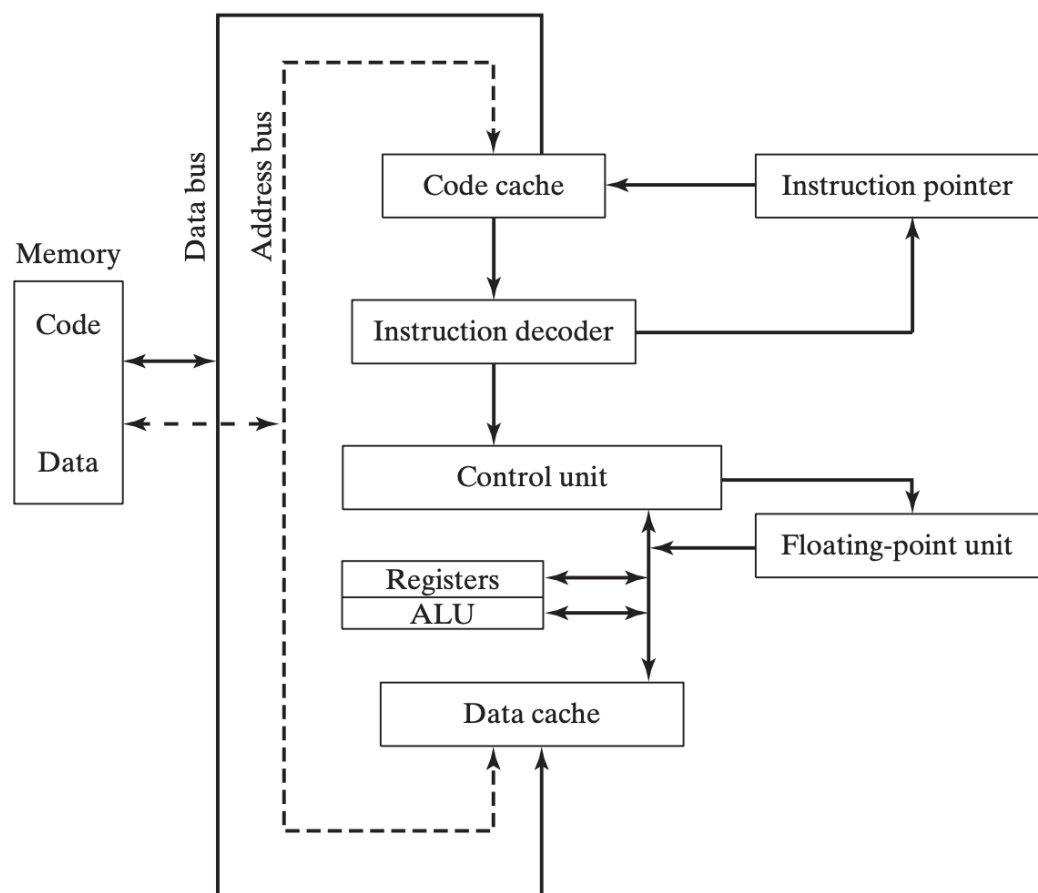
1. Tout d'abord, la CPU doit récupérer l'instruction dans une zone de mémoire appelée la file d'attente d'instructions. Juste après cela, il incrémente le pointeur d'instruction.
2. Ensuite, la CPU décode l'instruction en examinant son modèle de bits binaires. Ce modèle de bits peut révéler que l'instruction a des opérandes (valeurs d'entrée).
3. Si des opérandes sont impliqués, la CPU récupère les opérandes dans les registres et la mémoire. Parfois, cela implique des calculs d'adresse.
4. Ensuite, la CPU exécute l'instruction, en utilisant toutes les valeurs d'opérande qu'elle a récupérées au cours de l'étape précédente. Il met également à jour quelques indicateurs de statut (flags), tels que ZF, CF et OF.
5. Enfin, si un opérande de sortie faisait partie de l'instruction, la CPU stocke le résultat de son exécution dans l'opérande.

Nous simplifions généralement ce processus complexe en trois étapes de base : Récupérer, Décodez et exécutez.

Un opérande est une valeur qui est soit une entrée, soit une sortie vers une opération. Par exemple, l'expression $Z = X + Y$ a deux opérandes d'entrée (X et Y) et un seul opérande de sortie (Z).

Un schéma de principe montrant le flux de données dans une CPU typique est illustré à la Figure 2. Le diagramme permet de montrer les relations entre les composants qui interagissent pendant le cycle d'exécution des instructions.

1. Afin de lire les instructions du programme depuis la mémoire, une adresse est placée sur le bus d'adresses.
2. Ensuite, le contrôleur de mémoire place le code demandé sur le bus de données, rendant le code disponible dans le cache de code.
3. La valeur du pointeur d'instruction détermine quelle instruction sera exécutée ensuite. L'instruction est analysée par le décodeur d'instructions, provoquant les signaux numériques à envoyer à l'unité de contrôle, qui coordonne l'ALU et l'unité à virgule flottante. Bien que le bus de commande ne soit pas représenté sur cette figure, il transporte des signaux qui utilisent l'horloge système pour coordonner le transfert de données entre les différents composants de l'UC.



Les processeurs x86 ont trois modes de fonctionnement principaux: le mode protégé, le mode d'adresse réelle et le mode de gestion du système. Un sous-mode, nommé virtual-8086, est un cas particulier de mode protégé. Voici une brève description de chacun :

Mode protégé

Le mode protégé est l'état natif du processeur, dans lequel toutes les instructions et fonctionnalités sont disponibles. Les programmes reçoivent des zones de mémoire distinctes nommées segments et le processeur empêche les programmes de référencer la mémoire en dehors de leurs segments assignés.

Mode Virtual-8086 En mode protégé, le processeur peut exécuter directement des logiciels en mode adresse réelle tels que des programmes MS-DOS dans un environnement sûr. En d'autres termes, si un programme plante ou tente d'écrire des données dans la zone de mémoire système, cela n'affectera pas les autres programmes exécutés en même temps. Un système d'exploitation moderne peut exécuter plusieurs sessions virtual-8086 distinctes en même temps.

Mode adresse réelle

Le mode adresse réelle implémente l'environnement de programmation d'un ancien processeur Intel avec quelques fonctionnalités supplémentaires, telles que la possibilité de basculer dans d'autres modes. Ce mode est utile si un programme nécessite un accès direct à la mémoire système et aux périphériques matériels.

Mode de gestion du système

Le mode de gestion du système (SMM) fournit un système d'exploitation avec un mécanisme pour mettre en œuvre des fonctions telles que la gestion de l'alimentation et la sécurité du système. Ces fonctions sont généralement mises en œuvre par les fabricants d'ordinateurs qui personnalisent le processeur pour une configuration système particulière.

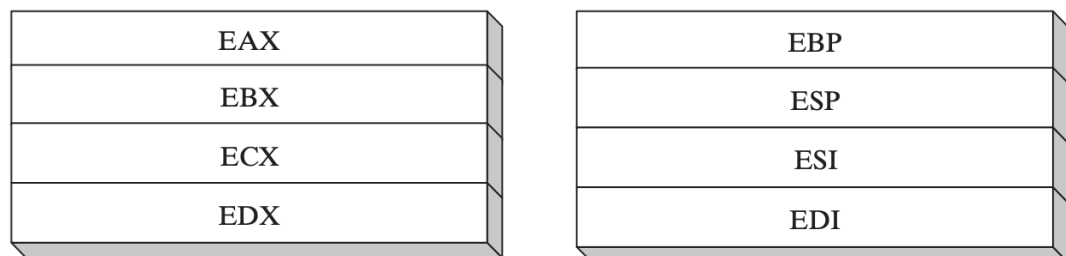
Les registres

Les registres sont des emplacements de stockage à haute vitesse directement à l'intérieur du processeur, conçus pour être accessibles à une vitesse beaucoup plus élevée que la mémoire conventionnelle.

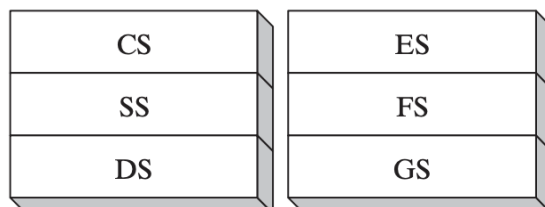
Lorsqu'une boucle de traitement est optimisée pour la vitesse, par exemple, les compteurs de boucle sont conservés dans des registres plutôt que dans des variables.

La figure 3 montre les registres d'exécution de programme de base. Il existe huit registres à usage général, six registres de segments, un registre d'indicateurs d'état du processeur (EFLAGS) et un pointeur d'instruction (EIP).

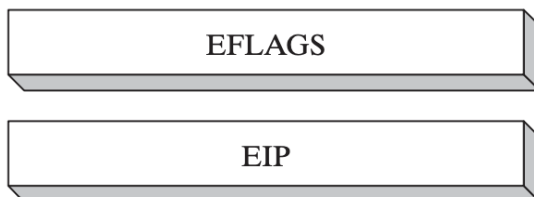
Registres généraux (32 bits)

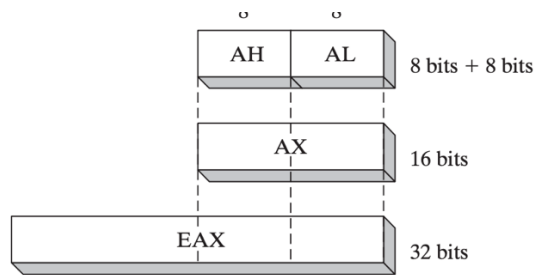


Les registres de segments (16 bits)



Deux autres registres (8 bits)





32-Bit	16-Bit	8-Bit (High)	8-Bit (Low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

32-Bit	16-Bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

Utilisations spécialisées

Certains registres à usage général ont des utilisations spécialisées :

- EAX est automatiquement utilisé par les instructions de multiplication et de division. Il est souvent appelé le registre d'accumulateur étendu.
- L'UC utilise automatiquement ECX comme compteur de boucle.
- ESP adresse les données sur la pile (une structure de mémoire système). Il est rarement utilisé pour les arithmétiques ou transfert de données. Il est souvent appelé registre de pointeur de pile étendue.
- ESI et EDI sont utilisés par les instructions de transfert de mémoire à grande vitesse. Ils sont parfois appelés l'index source étendu et les registres d'index de destination étendus.
- EBP est utilisé par les langages de haut niveau pour référencer les paramètres de fonction et les variables locales sur la pile. Il ne doit pas être utilisé pour l'arithmétique ordinaire ou le transfert de données, sauf à un niveau avancé de programmation. Il est souvent appelé registre de pointeur de trame étendu.

```
1: main PROC
2:     mov eax,5
3:     add eax,6
4:
5:     INVOKE ExitProcess,0
6: main ENDP
```