Introduction au développement logiciel
Notes de cours: assembleur

Georges-André Silber

Octobre 2025

Table des matières

Architectures externes et internes de processeurs

Une *architecture externe de processeur* ou *architecture de jeu d'instructions* – en anglais ISA, *Instruction Set Architecture* –, ou tout simplement *architecture de processeur*, est la spécification fonctionnelle d'un processeur, du point de vue du programmeur en langage machine. L'architecture comprend notamment le jeu d'instructions, les registres visibles par le programmeur, l'organisation de la mémoire et des entrées sorties, les modalités d'un éventuel support multiprocesseur.

Le terme *externe* permet de se différencier de la *microarchitecture* ou *architecture interne*, qui s'intéresse à l'implémentation pratique du comportement spécifié par une architecture externe. Une architecture externe donnée peut être implémentée sous forme de plusieurs microarchitectures. Ainsi, les sociétés Intel et AMD proposent toutes les deux des microprocesseurs d'architecture externe x86, mais avec des microarchitectures différentes.

Dans ce cours, nous ne nous interesserons qu'aux architectures externes de processeurs et emploierons indifféremment le terme *architecture* ou ISA pour la désigner. L'achitecture matérielle d'un ordinateur complet, incluant la microarchitecture d'un processeur, est un vaste sujet que nous n'aborderons dans ce cours hormis sous des aspects comportementaux nécessaires à une programmation système efficace.

Parmi toutes les architectures de processeurs existantes, nous ferons dans ce cours et dans les exercices un survol des architectures x86, ARM, qui couvrent la majorité des processeurs dans l'on trouve dans les machines actuelles.

Voici un exemple de fonction codée en langage C, ajoutant à un emplacement mémoire contenant un entier signé sur 64 bits (`long`), un entier signé sur 32 bits (`int`) :

```
void f(long *a, int c)
{
    *a = *a + c;
}
```

Voici le code équivalent en Rust :

```
fn f(a: &mut s64, c: s32) {
  let cst = c as s64;
  *a = *a + cst;
}
```

En assembleur x86 en 64 bits, le corps de la fonction peut être écrit ainsi, avec `a` contenu dans `rdi` et `b` contenu dans `esi`

```
movsxd rsi, esi            # rsi <- SignExtend(esi)
add qword ptr [rdi], rsi   # memory[rdi] <- memory[rdi] + rsi
```

En assembleur ARM en 64 bits (arm), on aurait :

```
ldr     x2, [x0]           # x2 <- memory[x0]
add     x1, x2, w1, sxtw   # x1 <- x2 + SignExtend(w1)  (sxtw = Sign eXTend Word)
str     x1, [x0]           # memory[x0] <- x1
```

Architecture x86

L'architecture x86 est une famille d'architectures dont le premier réprésentant matériel est le processeur 8086 d'Intel introduit en 1978. Architecture 16 bits à l'origine, elle a évolué en architecture 64 bits sous l'impulsion d'AMD en 2005[1].

Le processeurs implémentant cette architecture fonctionnent principalement dans deux modes :

— le mode en *adresses réelles*, dit aussi *real mode*, implémente l'environnement de programmation d'un processeur 8086 des origines, avec aujourd'hui des extensions, comme la possibilité d'utiliser les registres 32 bits ou la possibilité de passer en mode *protégé*. Le processeur démarre toujours dans ce mode lorsqu'il est alimenté en courant ;
— le mode *protégé*, qui est le mode natif du processeur – celui dans lequel il est censé fonctionner. C'est ce mode qui permet à un système d'exploitation d'être multi-tâches et d'avoir une représentation de la mémoire « *à plat* ».

L'architecture x86, adoptant une organisation des octets de type *little-endian* (voir fig. 1) possède notamment les registres suivants (voir figs. 2, 3, 4, 5, 6, 7, 8) :

— `eax` — Accumulator for operands and results data
— `ebx` — Pointer to data in the `ds` segment
— `ecx` — Counter for string and loop operations
— `edx` — I/O pointer
— `esi` — Pointer to data in the segment pointed to by the DS register ; source pointer for string operations
— `edi` — Pointer to data (or destination) in the segment pointed to by the `es` register ; destination pointer for string operations
— `esp` — Stack pointer (in the `ss` segment)
— `ebp` — Pointer to data on the stack (in the `ss` segment)

Les principales instructions sont présentées dans la documentation de référence Intel ou sur des sites comme `sandpile.org`. L'accès à la mémoire se fait selon des calculs d'adresse présentés à la fig.9. Les types de données fondamentaux que l'architecture x86 peut manipuler sont les entiers et les flottants, voir figs.10, 11, 12, 13, 14, 15, 16, 17. Le processeur a plusieurs modes de protection, voir fig. 18.

Architecture ARM

ARM[2] est une famille d'architectures RISC de microprocesseurs, avec des variantes pour plusieurs environnements, développées par *ARM Ltd* depuis 1983.

La société *Arm Ltd.*, à l'origine une coentreprise fondée par Acorn Computers, Apple Computers et VLSI Technologies, développe les architectures et concède des licences à d'autre compagnies, qui conçoivent leurs propres produits implémentant ces architectures, comme des SoC intégrant plusieurs composants : processeurs,

1. Intel avait à l'époque tablé sur une nouvelle architecture aujourd'hui quasiment disparue, l'Itanium.
2. stylisé en bas de casse en arm, auparavant un acronyme de *Advanced RISC Machines*, et à l'origine de *Acorn RISC Machine*

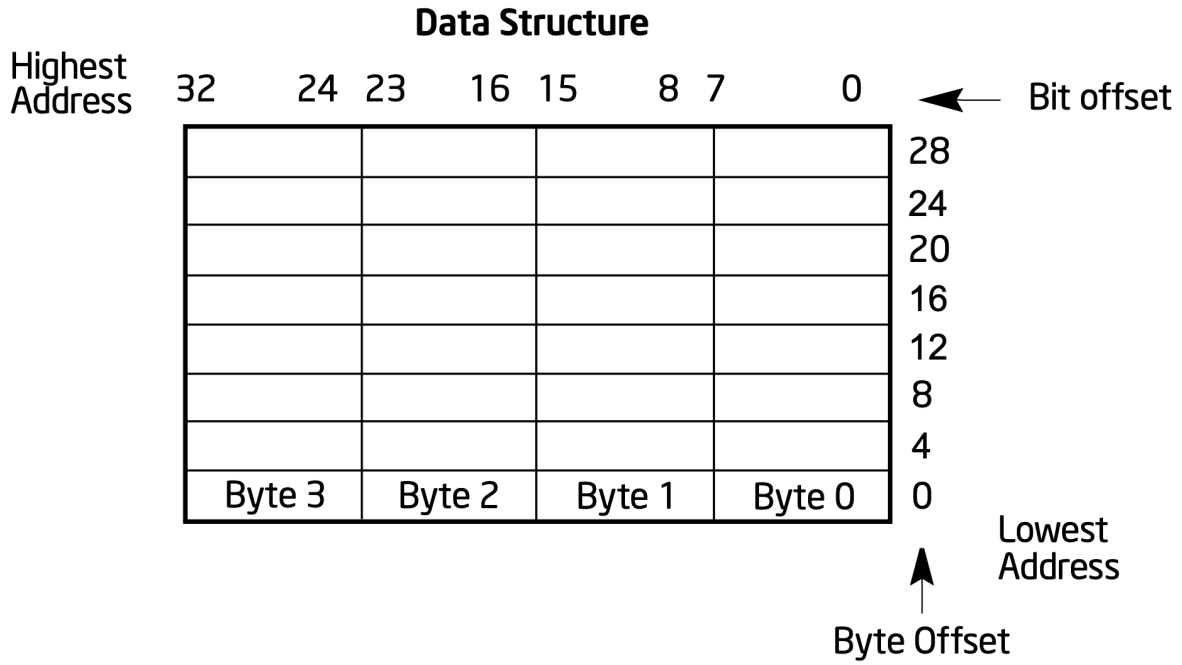

FIGURE 1 – Bits et octets, architecture Intel *little-endian.*

mémoire, GPU. Elle conçoit également des cœurs qui implémentent ces architectures et concède des licences à des compagnies qui intègrent ces conceptions à leurs propres produits. Ainsi, *Arm Ltd.* conçoit des architectures externes et internes sans fabriquer elle-même de composants électroniques, elle vend uniquement de la propriété intellectuelle (*Intellectual Properties*).

Il y a eu au cours du temps plusieurs générations d'architectures ARM, l'ARM1 utilisait une structure interne de 32 bits avec un espace d'adressage sur 26 bits, limitant la mémoire principale à 64 Mo, architecture évoluant en plusieurs étapes pour avoir un espace d'adressage sur 32 bits, puis passer à une architecture en 64 bits à partir de l'ARMv8 en 2011. Plusieurs jeux d'instructions additionnels ont été introduits, comme l'extension Thumb avec des instructions en 32 et 16 bits permettant d'améliorer la densité du code ou encore Jazelle pour prendre en charge au niveau matériel le bytecode Java et JavaScript.

Dotés d'une architecture relativement plus simple que d'autres familles de processeurs et faibles consommateurs d'électricité, les processeurs ARM sont aujourd'hui dominants dans le domaine de l'informatique embarquée, en particulier la téléphonie mobile et les tablettes. De plus, les processeurs ARM sont aujourd'hui de plus en plus utilisés sur les postes de travail et les serveurs, comme par exemple les ordinateurs Apple se fondant sur des puces M1 ou les serveurs à base de Neoverse. Le supercalculateur actuellement le plus puissant du monde utilise une architecture de processeur ARMv8.2-A, fabriquée par Fujitsu avec 48 cœurs : le supercalculateur utilise 158 976 processeurs, soit 7 630 848 cœurs. [3]

Le Raspberry Pi 4 B comporte un processeur à l'architecture ARMv!. [4]

3. https ://www.r-ccs.riken.jp/en/fugaku/about/
4. SoC Broadcom BCM2711, Quad core Cortex-A72 1.5 GHz, voir `https://developer.arm.com/ip-products/processors/cortex-a/cortex-a72` et `https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-datasheet.pdf`

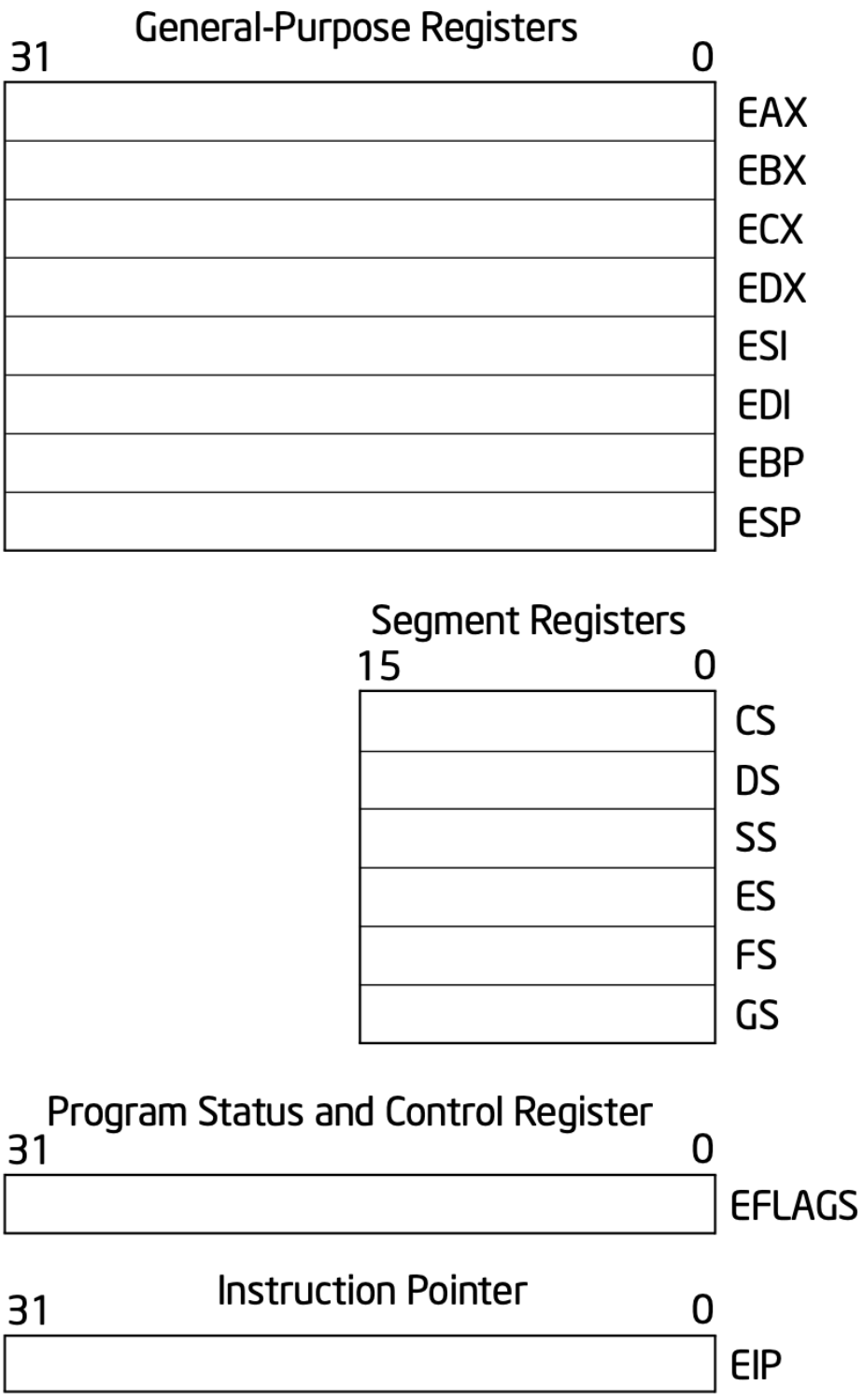

FIGURE 2 – Registres généraux et de segment, x86 en 32 bits

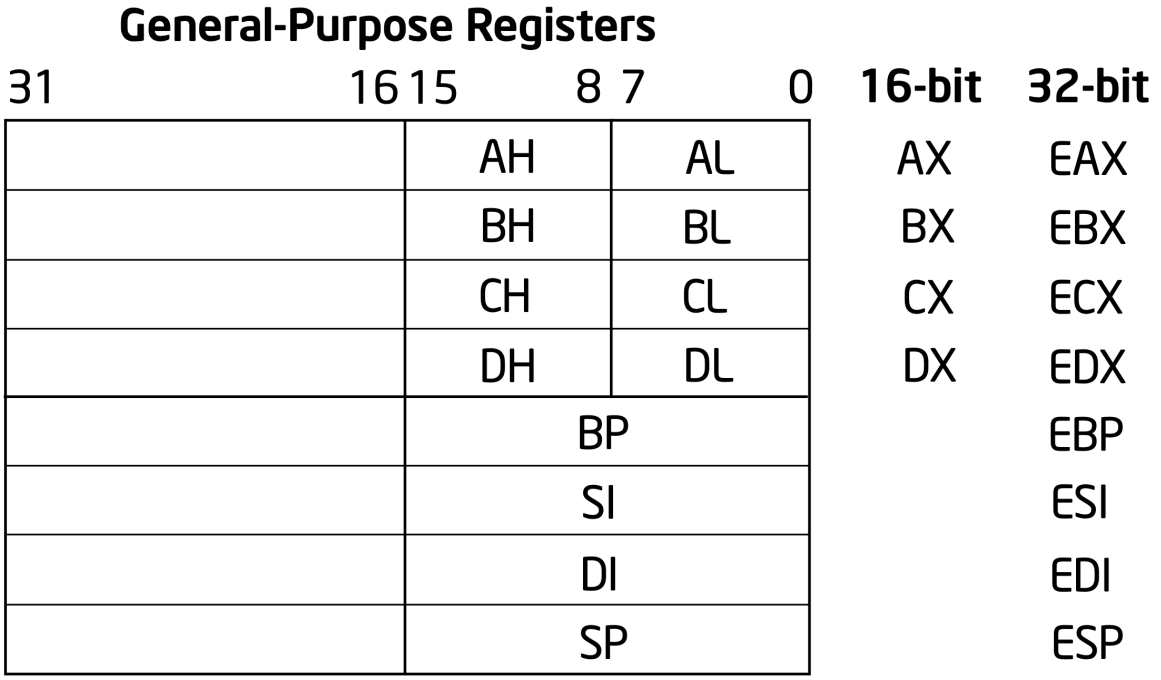

FIGURE 3 – Sous-registres, x86 en 32 bits

Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8B - R15B
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

FIGURE 4 – Registres x86 en 64 bits

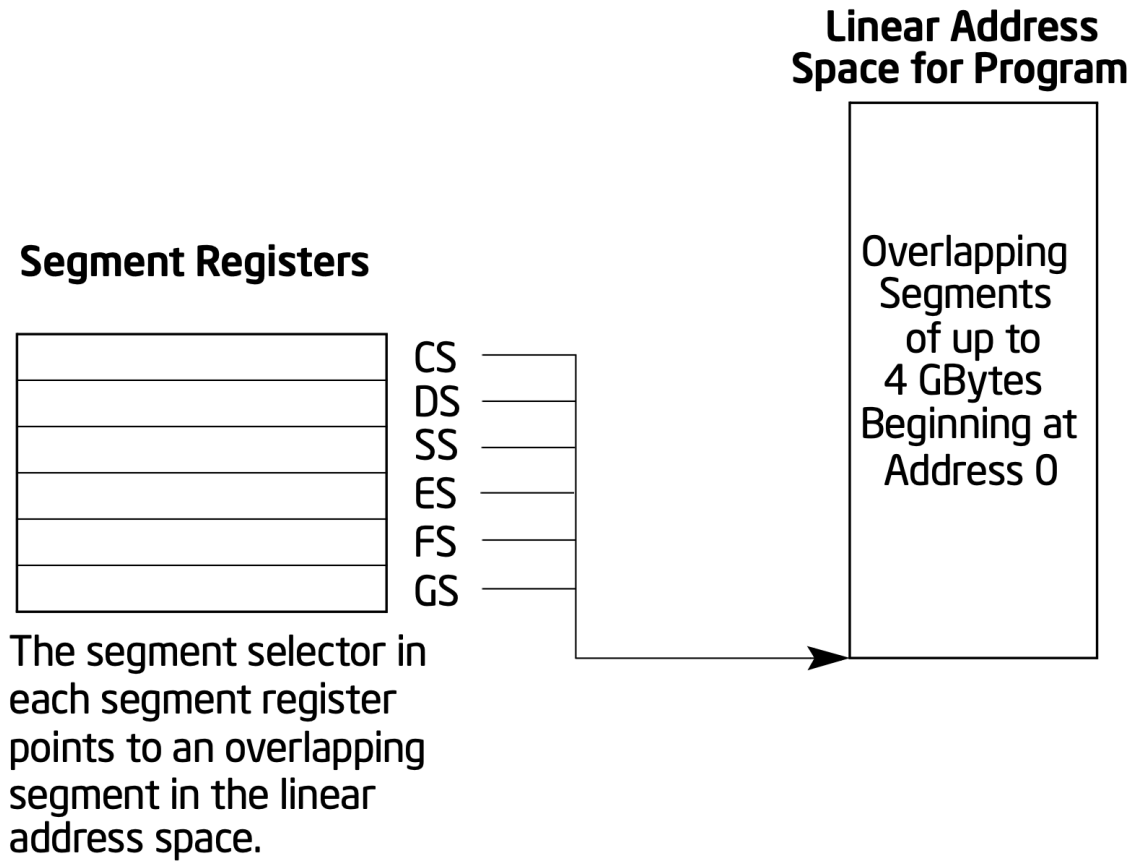

FIGURE 5 – Registres de segment, mémoire segmentée

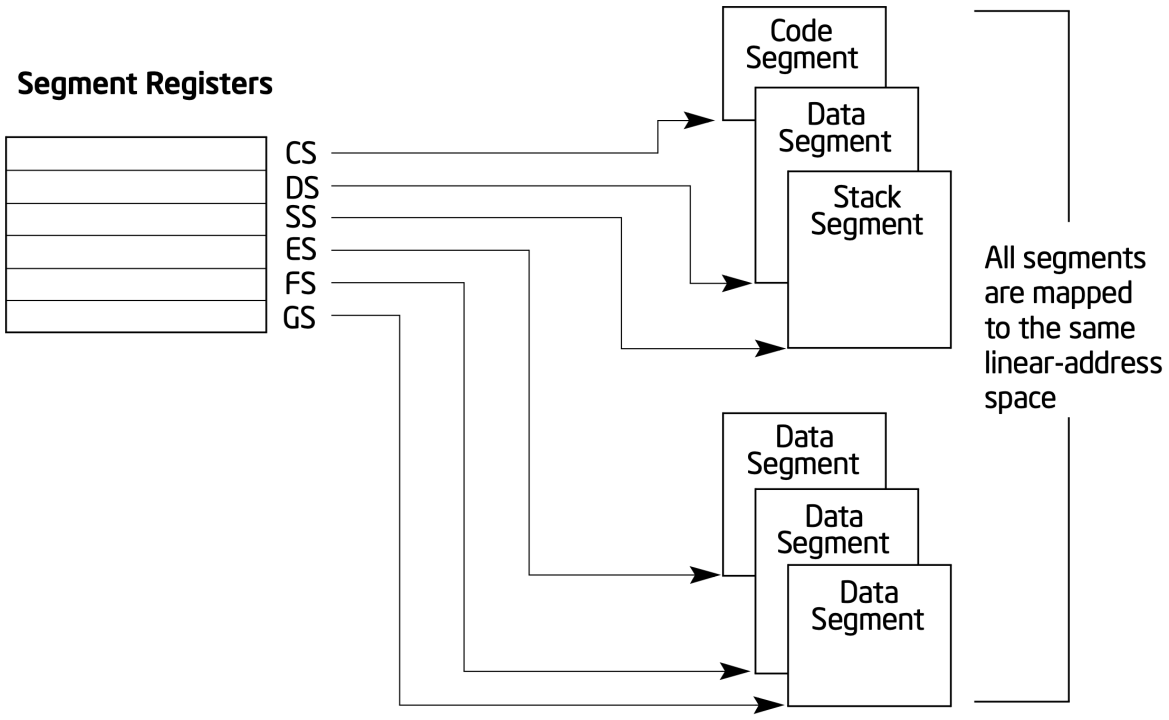

FIGURE 6 – Registres de segment, mémoire à plat

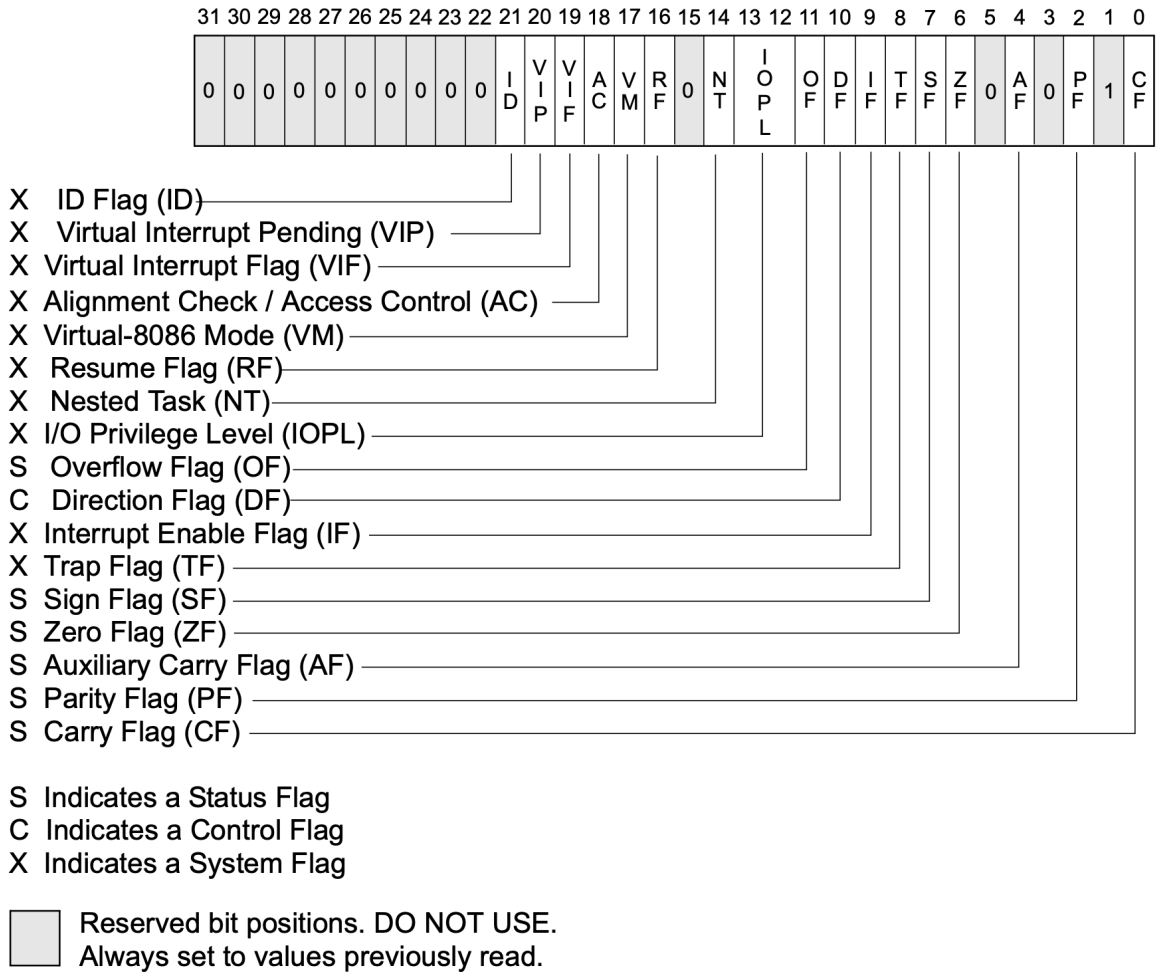

FIGURE 7 – Registre d'état

FIGURE 8 – Tables des registres de l'architecture x86

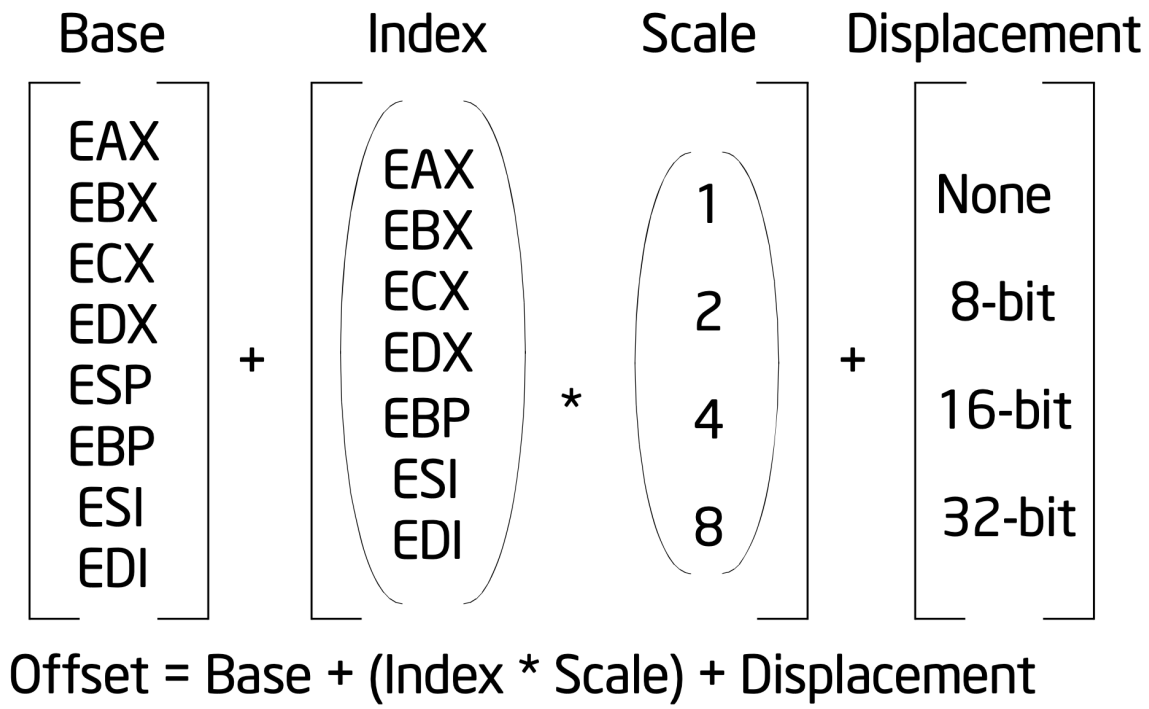

FIGURE 9 – Calcul d'adresse effective

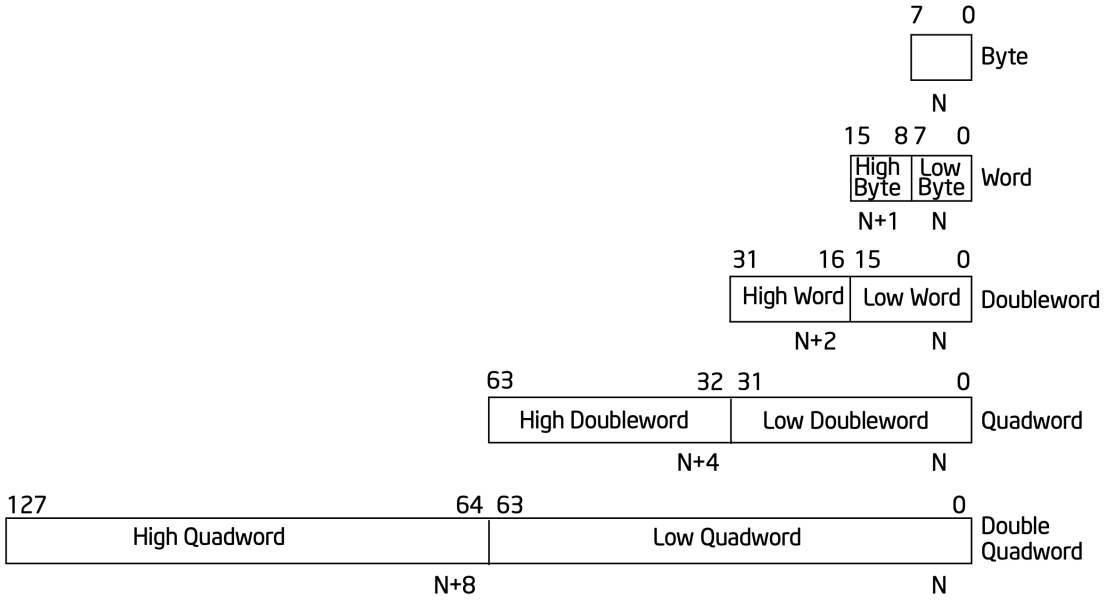

FIGURE 10 – Fundamental datatypes

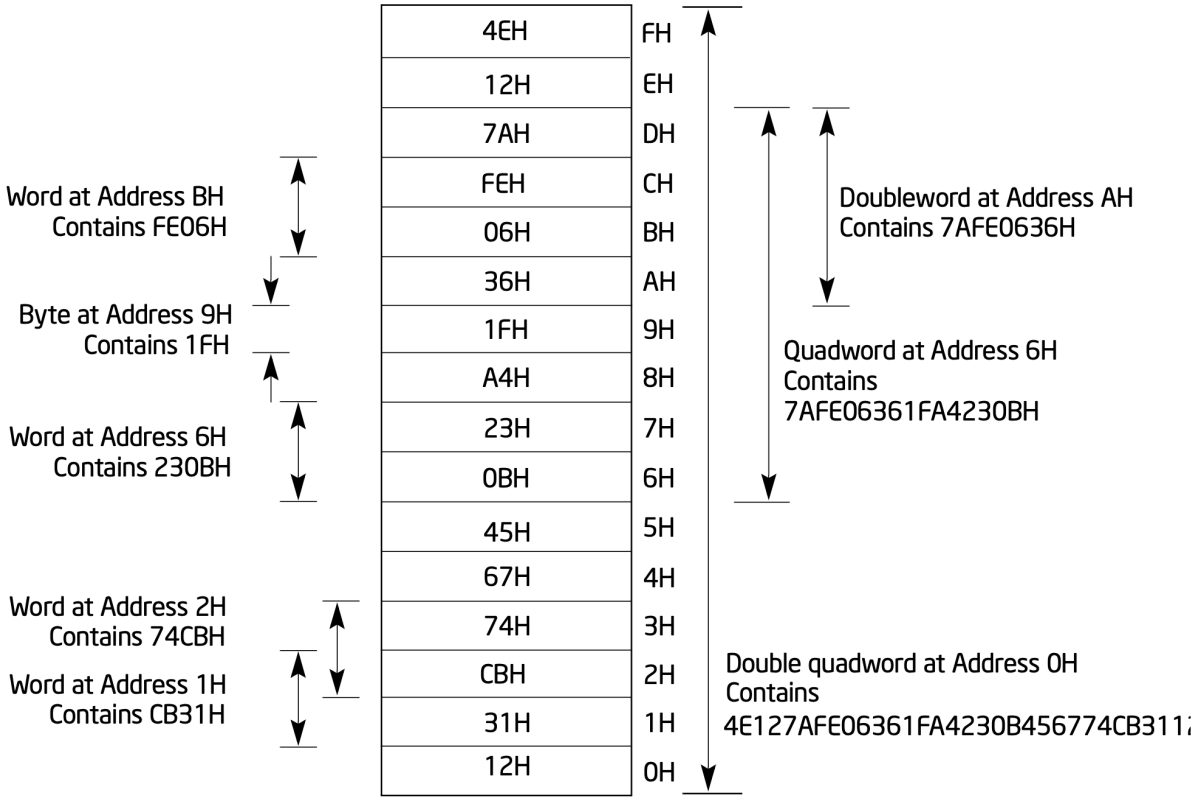

FIGURE 11 – Word in memory

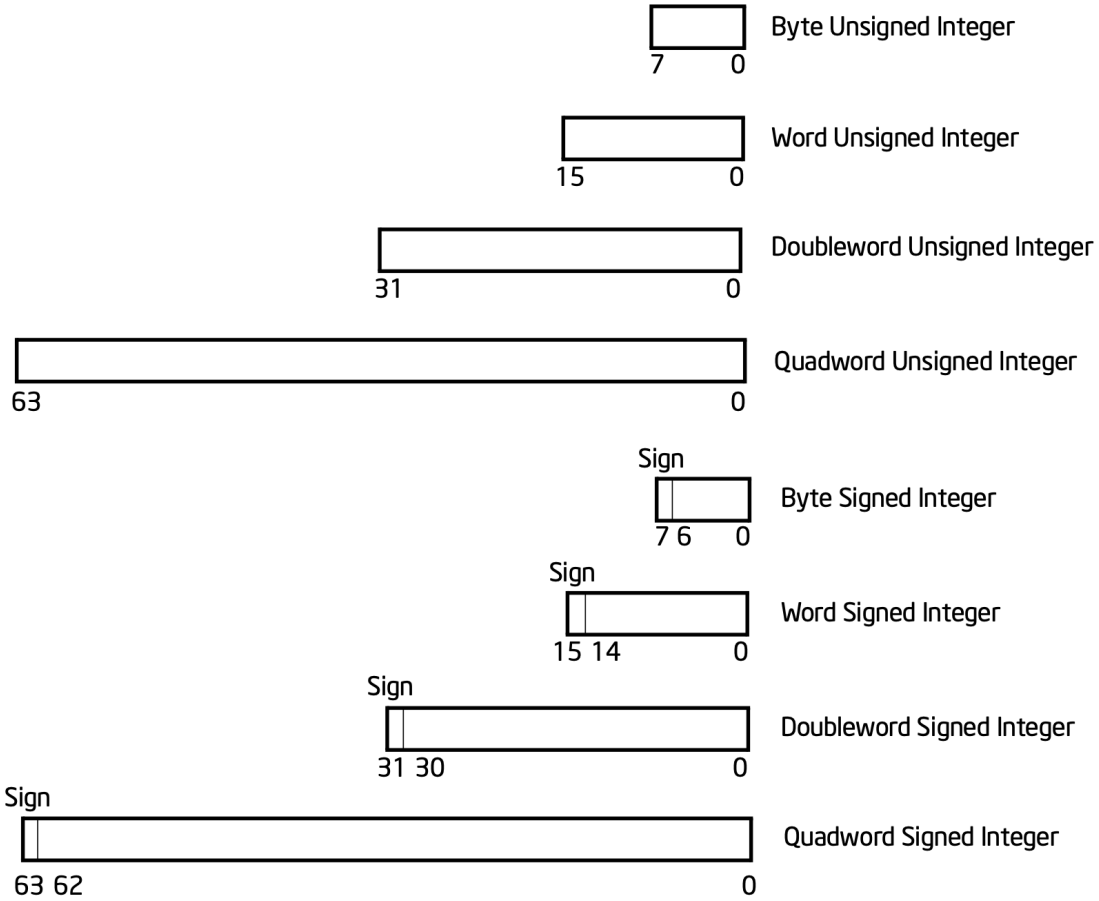

FIGURE 12 – Integers

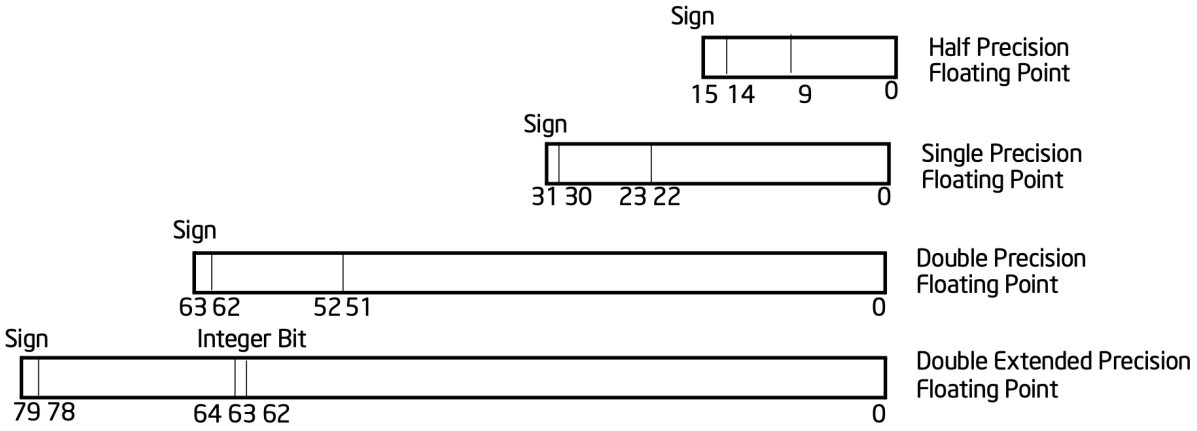

FIGURE 13 – Floats

Class		Two's Complement Encoding	
		Sign	
Positive	Largest	0	11..11
		.	.
		.	.
	Smallest	0	00..01
Zero		0	00..00
Negative	Smallest	1	11..11
		.	.
		.	.
	Largest	1	00..00
Integer indefinite		1	00..00
		Signed Byte Integer: Signed Word Integer: Signed Doubleword Integer: Signed Quadword Integer:	← 7 bits → ← 15 bits → ← 31 bits → ← 63 bits →

FIGURE 14 – Représentation des entiers signés : le complément à deux

Data Type	Length	Precision (Bits)	Approximate Normalized Range	
			Binary	Decimal
Half Precision	16	11	2^{-14} to 2^{15}	3.1×10^{-5} to 6.50×10^{4}
Single Precision	32	24	2^{-126} to 2^{127}	1.18×10^{-38} to 3.40×10^{38}
Double Precision	64	53	2^{-1022} to 2^{1023}	2.23×10^{-308} to 1.79×10^{308}
Double Extended Precision	80	64	2^{-16382} to 2^{16383}	3.37×10^{-4932} to 1.18×10^{4932}

FIGURE 15 – Représentation des réels : intervalles des représentations à virgule flottante

Class		Sign	Biased Exponent	Significand	
				Integer[1]	Fraction
Positive	+∞	0	11..11	1	00..00
	+Normals	0 . . 0	11..10 . . 00..01	1 . . 1	11..11 . . 00..00
	+Denormals	0 . . 0	00..00 . . 00..00	0 . . 0	11.11 . . 00..01
	+Zero	0	00..00	0	00..00
Negative	–Zero	1	00..00	0	00..00
	–Denormals	1 . . 1	00..00 . . 00..00	0 . . 0	00..01 . . 11..11
	–Normals	1 . . 1	00..01 . . 11..10	1 . . 1	00..00 . . 11..11
	-∞	1	11..11	1	00..00

FIGURE 16 – Représentation des réels : codage à virgule flottante

Class		Sign	Biased Exponent	Significand	
				Integer[1]	Fraction
NaNs	SNaN	X	11..11	1	0X..XX[2]
	QNaN	X	11..11	1	1X..XX
	QNaN Floating-Point Indefinite	1	11..11	1	10..00
	Half-Precision Single-Precision: Double-Precision: Double Extended-Precision:		← 5Bits → ← 8 Bits → ← 11 Bits → ← 15 Bits →		← 10 Bits → ← 23 Bits → ← 52 Bits → ← 63 Bits →

FIGURE 17 – Représentation des réels : codage à virgule flottante (suite)

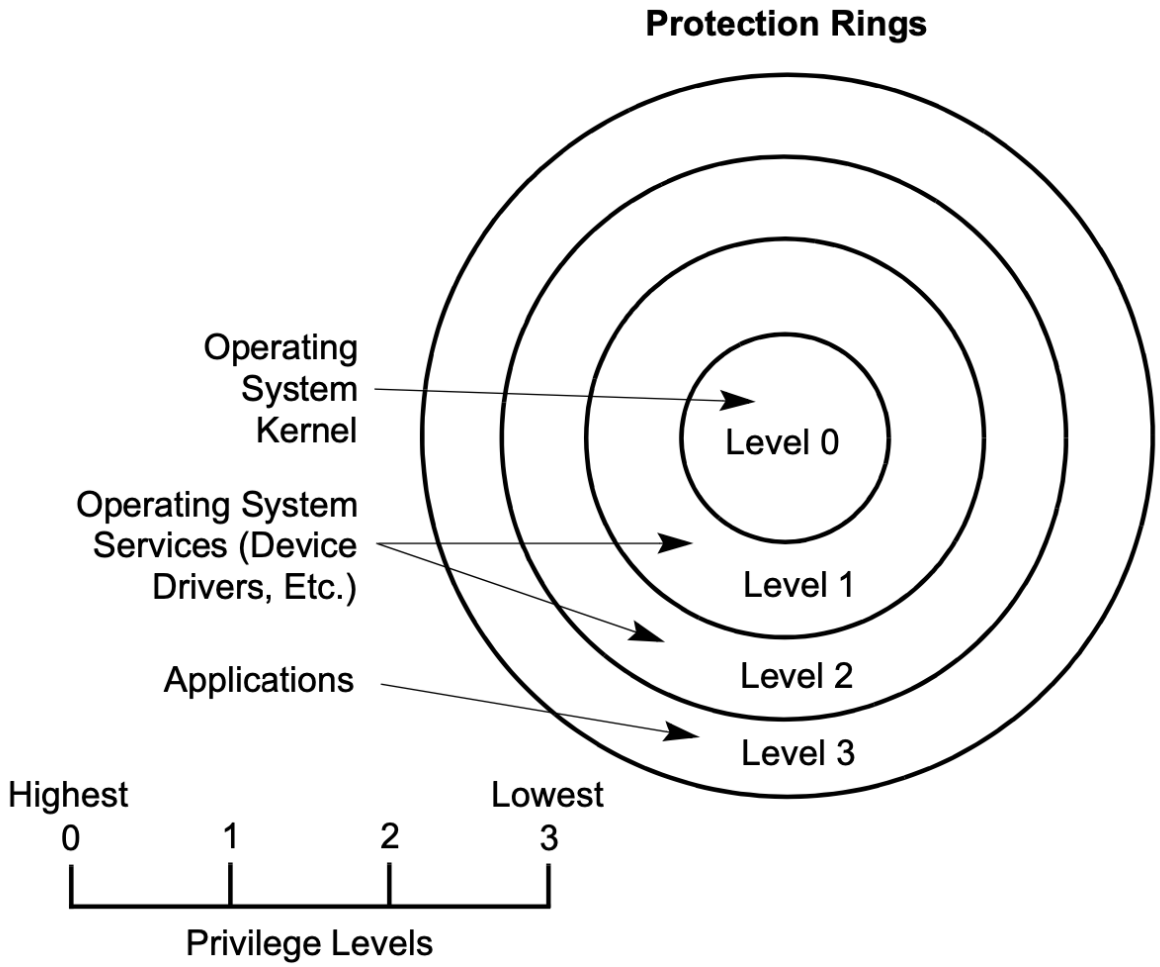

FIGURE 18 – Protection rings