

Module M211

Cours Architecture

Programmation des mécanismes de base d'un système
informatique

Marie-Agnès Peraldi-Frati
MCF Informatique

Objectif des modules ASR

- ▶ Le champ disciplinaire « **Architecture matérielle - Systèmes d'exploitation - Réseaux** » concerne à la fois les connaissances de base sur le matériel (codage de l'information, fonctionnement interne des ordinateurs), les systèmes d'exploitation professionnels multitâches et multi utilisateurs (utilisation, administration, utilisation des services par programmation), ainsi que les réseaux et leurs normes (organisation, fonctionnement, éléments d'administration, techniques de programmation d'applications réparties ou distribuées).

M211: Organisation et périmètre

► Organisation :

- Architecture et programmation des mécanismes de base d'un système informatique.
- 1h Cours, 3h TD/TP par semaine pendant 5 semaines.
- 1h d'examen en fin de module
- Utilisation de l'émulateur Proc X86 en TP <http://www.emu8086.com>

► Périmètre

- De quoi est composé un ordinateur ?
- Quel est le lien entre le matériel et le logiciel?
 - Modèles d'exécution (séquentiel, parallèle données, parallèles instructions)
 - Modèle de mémoires (registres, cache L1 L2 ...)
- Comment s'exécute un programme ?
- Comment fonctionnent les divers périphériques ?

Agenda du cours

- ▶ Cours 1 :
 - Généralités archi/assembleur
 - Manipulation émulateur
 - **Code, UAL, registres, mémoire**
 - Exécution, visualisation registres
- ▶ Cours 2 :
 - Différents **types de mémoires**
 - Accès mémoire (code, données, E/S)
 - Manipulation structure de données en assembleur
- ▶ Cours 3 :
 - **Notion de Pile**
 - Appel de procédures
- ▶ Cours 4 :
 - **Interruptions**
 - E/S
- ▶ Cours 5 :
 - Développement programme
 - E/S , IT,...
- ▶ Cours 6 :
 - **Examen**

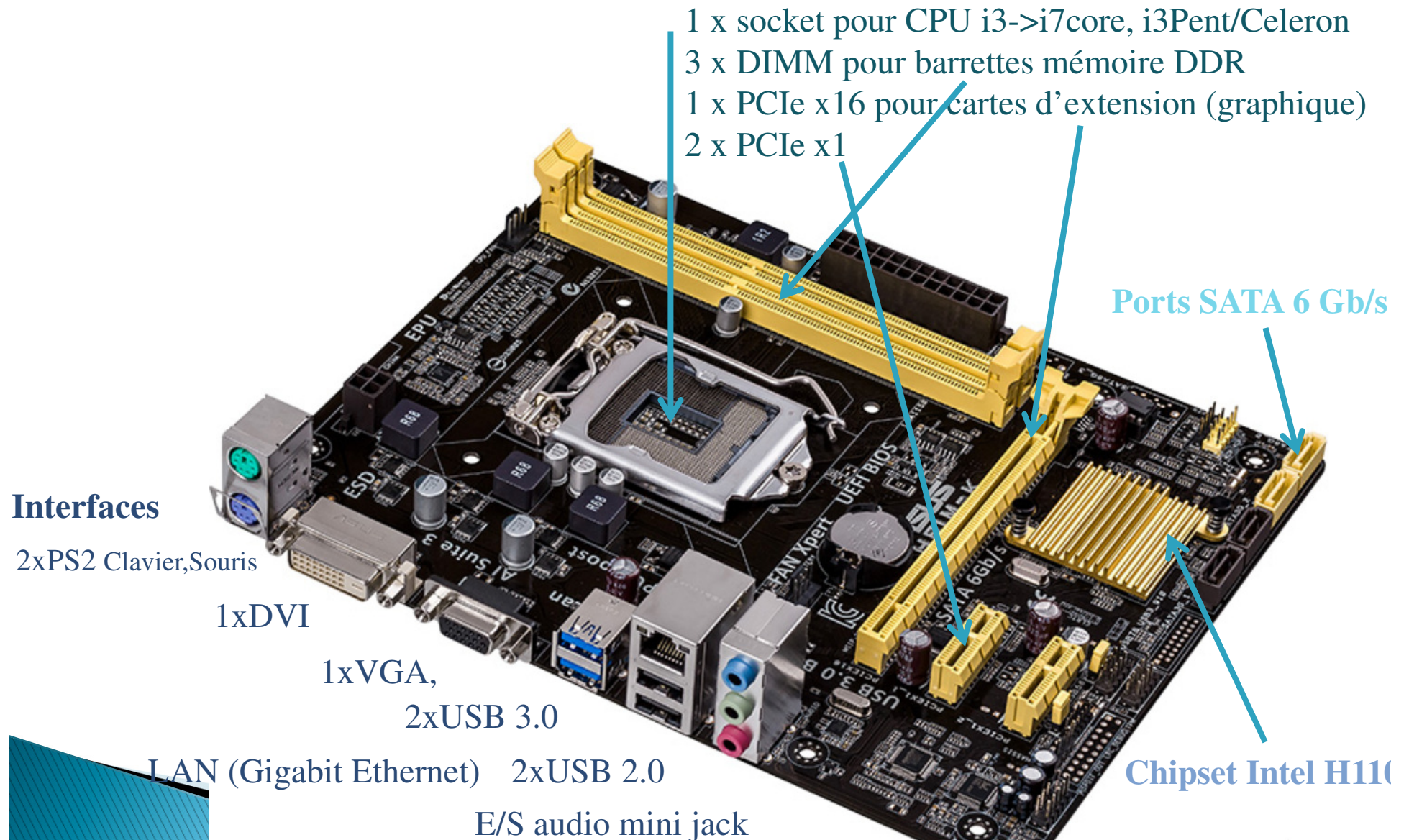
PPN National

UE21	Informatique approfondie	Volume Horaire 8h CM, 10h TD, 12h TP
	Architecture matérielle - Systèmes d'exploitation - Réseaux	
M2101	Architecture et Programmation des mécanismes de base d'un système informatique	Semestre 2
<u>Objectifs du module :</u> <ul style="list-style-type: none"> Savoir développer des applications simples mettant en œuvre les mécanismes de bas niveau d'un système informatique 		
<u>Compétences visées :</u> <i>Compétences citées dans le Référentiel d'activités et de compétences pour les activités :</i> <ul style="list-style-type: none"> FA2-A : Administration de systèmes, de logiciels et de réseaux FA2-B : Conseil et assistance technique à des utilisateurs, clients, services FA1-B : Conception technique d'une solution informatique 		
<u>Prérequis :</u> <ul style="list-style-type: none"> M1101 M1102 		
<u>Contenus :</u> <ul style="list-style-type: none"> Langages de programmation bas niveau Mécanismes de bas niveau d'un système informatique Etude d'un système à microprocesseur ou microcontrôleur (réel ou simulé) avec ses composants (mémoires, interfaces, périphériques...) 		
<u>Modalités de mise en œuvre :</u> <ul style="list-style-type: none"> Utilisation du langage C et/ou d'un langage d'assemblage (assembleur) Observation de l'exécution pas à pas d'un programme à l'aide d'un outil de simulation/déverminage d'un processeur simple Développement de programmes simples permettant d'illustrer les principaux mécanismes de bas niveau d'un système informatique Etude des mécanismes de gestion des interruptions 		
<u>Prolongements possibles :</u> <ul style="list-style-type: none"> Programmation des Systèmes Embarqués Processus de compilation Etude du fonctionnement d'un système d'exploitation (OS : <i>Operating System</i>) minimal embarqué 		

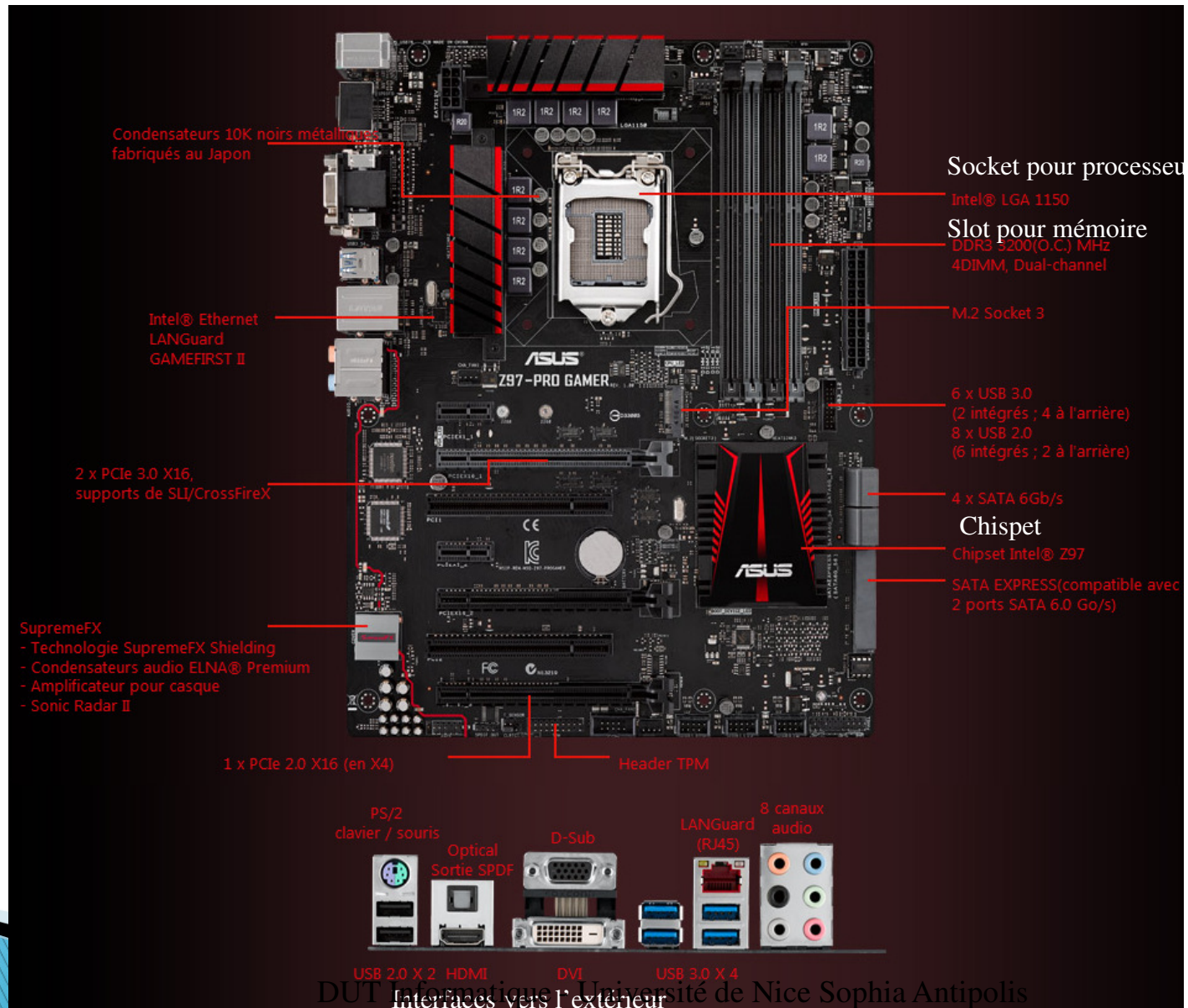
Un ordinateur

- ▶ Unité centrale
 - Carte mère
 - Processeur
 - Mémoire vive /mémoire de stockage
 - Chipset
 - Carte graphique
 - Alimentation
- ▶ Périphériques d'entrée : clavier, souris
- ▶ Périphériques de sortie : Ecrans HP
- ▶ Périphériques de stockage disques, lecteurs CD, DVD

Carte mère ASUS H61M-K



Exemple Carte mère ATX ASUS pro Gamer



MotherBoard inside !

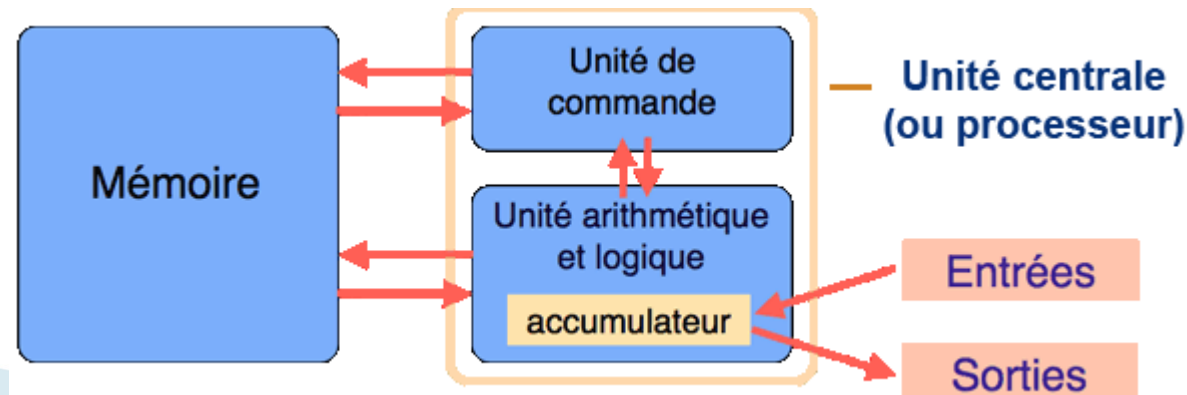
- ▶ CPU = Processeur (Central Processing Unit)
 - Qui exécute les instructions de votre programme
- ▶ Processeurs dédiés présents sur une carte spécifique ou intégrés sur une carte-mère
 - GPU = (Graphics Processing Unit) assure les fonctions de calcul de l'affichage
 - PPU = (Physics Processing Unit) calcul de la dynamique physique (fluides, structures complexes ...)

CPU inside !

- ▶ Processeur : Circuit électronique cadencé par une horloge interne (quartz)
- ▶ **Fréquence d'horloge (cycle)** = Nbre d'impulsion de quartz /secondes (Hertz)
- ▶ A chaque impulsion d'horloge le processeur va lire une instruction stockée dans un registre d'instruction et l'exécute.
- ▶ Une **instruction** s'exécute en un ou plusieurs cycles
- ▶ **Puissance** du processeur = Nbre d'instructions par seconde. **MIPS** (Millions Instruction par seconde)

Architecture logique d'un processeur Von Neumann(1945)

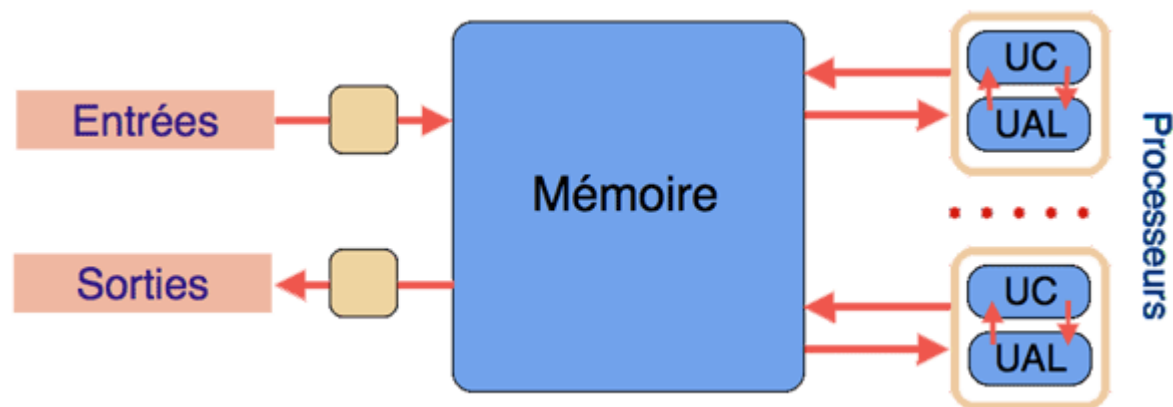
- **Unité de commande (UC)**, qui organise le séquençement des instructions
 - **Compteur ordinal** contient l'adresse de l'instruction en cours d'exécution, (*IP*)
 - soit **incrémenté**
 - soit **modifié** lors des sauts
 - **Registre d'état** : ensemble de flags qui donne l'état du calcul (*AF CF OF SF PF ZF DF IF TF*)
- **Unité arithmétique UAL**, exécute les instructions arithmétiques et logiques
 - **Accumulateur** (ou registre) : mémorise les résultats intermédiaires des calculs avant E/S (*AX BX CX DX*)
- **Mémoire** peut contenir les programmes et les données
 - **Programme** : suite d'instructions codées selon un format conventionnel (**jeu d'instruction**) (*CS*)
 - **Données** : valeurs encodées selon un format conventionnel (entier, chaîne, flottants ...) (*DS*)



Architecture logique d'un processeur

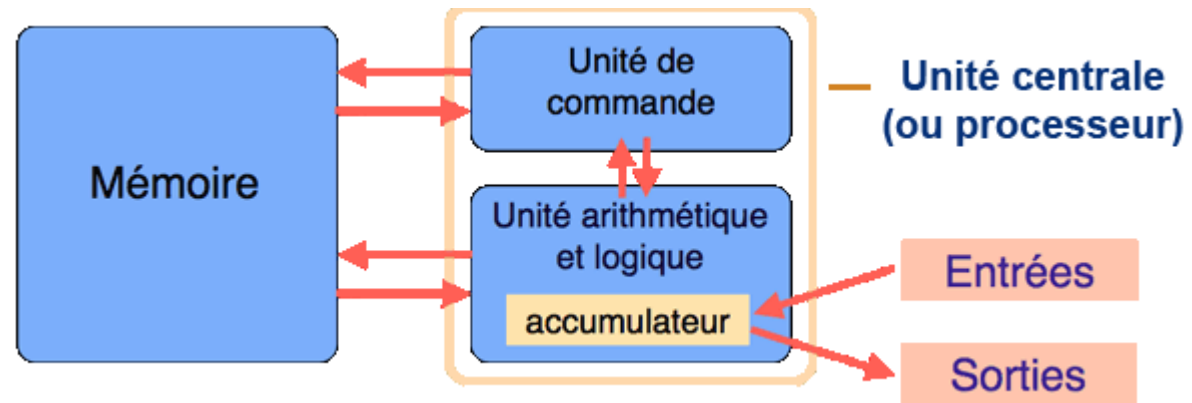
Evolutions de AVN à nos jours

- **Processeurs multiples**, qu'il s'agisse d'unités séparées ou de « cœurs » multiples. cela permet d'atteindre des puissances de calcul élevées non atteignable sur un seul processeur
- Les **entrées-sorties**, initialement commandées par l'unité centrale, sont désormais sous le contrôle de processeurs autonomes (canaux d'entrée-sortie et mécanismes assimilés).
- **Multiprogrammation**: OS et temps partagé entre plusieurs programmes,



Revenons à l'archi de Von Neumann

- **Unité de commande (UC)**, qui organise le séquençement des **instructions**
 - **Compteur ordinal** contient l'adresse de l'instruction en cours d'exécution,
 - soit **incrémenté**
 - **modifié** lors des sauts
 - **Registre d'état**
- **Unité arithmétique UAL**, exécute les instructions arithmétiques et logiques
- **Accumulateur** (ou registres) : mémorise les résultats intermédiaires des calculs avant **E/S**
- **Mémoire** peut contenir les programmes et les données
 - **Programme** : suite d'instructions codées selon un format conventionnel (**jeu d'instruction**)
 - **Données** : valeurs encodées selon un format conventionnel (entier, chaîne, flottants ...)



Une instruction c'est quoi ?

► En C

```
#include <stdio.h>
int a=1;
int b=2;
void main(){
    b=15; /* Ceci est un instruction C d'affectation */
    a=b+10;
}
```

► En Assembleur

```
DONNEES      SEGMENT
               intA          Db          1
               intB          Db          2
DONNEES      ENDS
CODE         SEGMENT
MAIN        PROC          FAR
    ; L'affectation dans une var nécessite 2 instructions en
    ASM
    mov Al, 15 ; hex=0fh or bin=00001111b
    mov intB, Al
    ; intA reçoit intB+10
    mov Bl, 10 ; ; hex=0ah or bin=00001010b
    add Al,B1 ; Al <- Al+B1
    mov intA,Al
    RET
MAIN        ENDP
CODE        ENDS
```

Une instruction c'est quoi ?

► En code machine 80x86

Dans le jeu d'instruction assembleur, à chaque instruction ASM correspond un code machine.

Exemple : MOV AL, 0Fh donne : **B0 0F**

Avec **B0** le code signifiant « déplacer une valeur dans AL » et 0F étant cette valeur.

Le code machine qui correspond à b=15; est stocké dans la RAM et vaut **B0 0F A2 01**

Code assembleur

Variable b

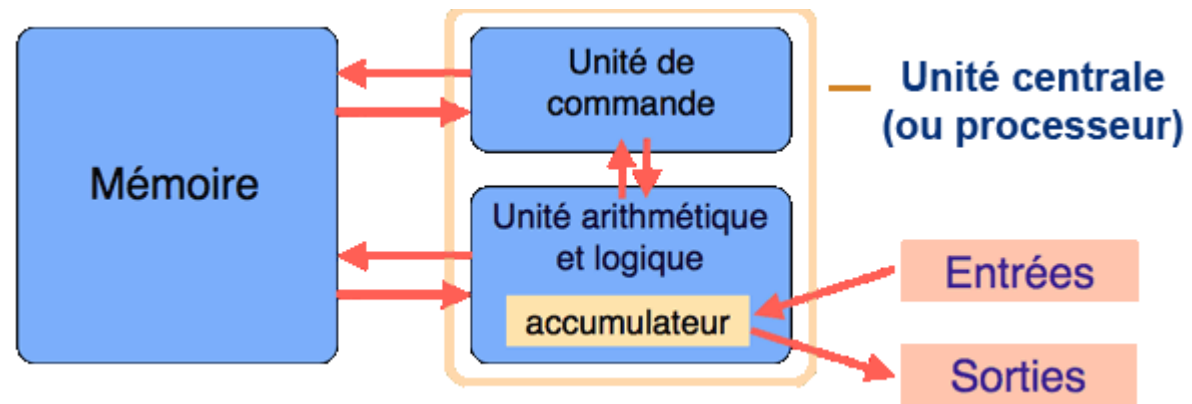
Code machine

The screenshot displays an 80x86 emulator interface with four main panels:

- original source code:** Shows assembly code. Line 09, `mov AL, 15 ; hex=0fh or bin=00`, is highlighted in yellow. Line 13, `add AL, BL ; AL <- AL+BL`, is highlighted in green.
- variables:** A table showing the value of variable `INTB` as `02h`.
- emulator: TP1_Somme2.bin:** Shows the state of registers and memory.
 - registers:** AX=0000, BX=0000, CX=0000, DX=0000, CS=0100, IP=0010, SS=0100, SP=FFFE, BP=0000, SI=0000, DI=0000, DS=0100, ES=0100.
 - memory dump:** Shows the instruction `MOV AL, 0Fh` at address `0100:0010` with hex value `B0 0F` and decimal value `176`.
- Random Access Memory:** A table showing the memory layout. The instruction `B0 0F A2 01` is highlighted in green at address `0100:0010`.

Revenons à l'archi de Von Neumann

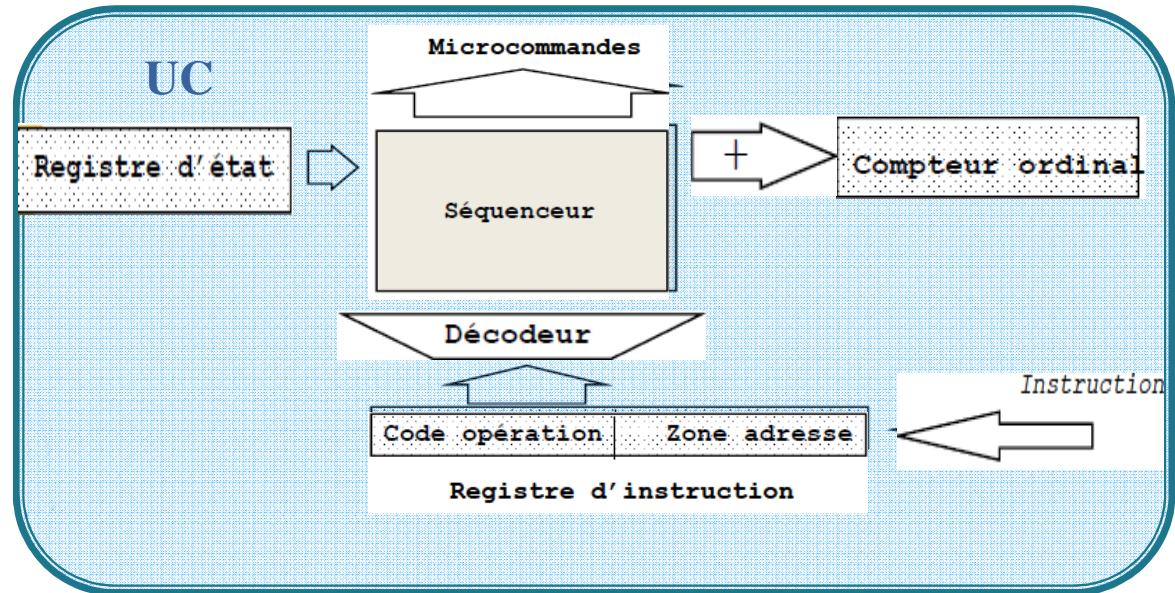
- **Unité de commande (UC)**, qui organise le séquençement des instructions
 - **Compteur ordinal** contient l'adresse de l'instruction en cours d'exécution,
 - soit **incrémenté**
 - **modifié** lors des sauts
 - **Registre d'état**
- **Unité arithmétique UAL**, exécute les instructions arithmétiques et logiques
- **Accumulateur** (ou registre) : mémorise les résultats intermédiaires des calculs avant E/S
- **Mémoire** peut contenir les programmes et les données
 - **Programme** : suite d'instructions codées selon un format conventionnel (**jeu d'instruction**)
 - **Données** : valeurs encodées selon un format conventionnel (entier, chaîne, flottants ...)



Unité de commande :

organise l'exécution des instructions

- ▶ Le **compteur Ordinal**
 - contient l'adresse mémoire de la prochaine instruction
 - Au chargement d'un programme par l'OS le CO récupère l'adresse mémoire de la première instruction du programme.
- ▶ Le séquenceur va à la **recherche en mémoire** de l'instruction
- ▶ L'Instruction stockée dans le **registre d'instruction**
- ▶ Elle est décodée par le **décodeur**
- ▶ les **registres d'état** sont lus
- ▶ Le **séquenceur** génère les microcommandes pour exécuter l'instruction (vers ALU, vers mémoire ...)
- ▶ A la fin de l'instruction le **CO** est incrémenté pour passer à l'instruction suivante



Unité de commande :

organise l'exécution des instructions

Compteur
Ordinal

Registres d'état

The screenshot shows a debugger interface with three main panels. On the left, the 'registers' panel lists 16 registers (AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, DS, ES) with their current values. The 'IP' register is highlighted with a blue box. Below it, the 'flags' panel shows various status flags (CF, ZF, SF, OF, PF, AF, IF, DF) with their current states. The 'IF' flag is highlighted with a red box. The main panel is divided into two sections: a memory dump on the left and a disassembled instruction list on the right. The memory dump shows addresses from 01000 to 01020 with corresponding hex values and ASCII characters. The instruction list shows assembly code for the same addresses. The instruction at address 01010, 'MOV AL, 0Fh', is highlighted with a blue box. The instruction at address 01011, 'MOV [00001h], AL', is highlighted with a green box. The instruction at address 01012, 'MOV BL, 0Ah', is highlighted with a green box. The instruction at address 01013, 'ADD AL, BL', is highlighted with a green box. The instruction at address 01014, 'MOV [00000h], AL', is highlighted with a green box. The instruction at address 01015, 'INT 016h', is highlighted with a green box. The instruction at address 01016, 'NOP', is highlighted with a green box. The instruction at address 01017, 'NOP', is highlighted with a green box. The instruction at address 01018, 'NOP', is highlighted with a green box. The instruction at address 01019, 'NOP', is highlighted with a green box. The instruction at address 0101A, 'NOP', is highlighted with a green box. The instruction at address 0101B, 'NOP', is highlighted with a green box. The instruction at address 0101C, 'NOP', is highlighted with a green box. The instruction at address 0101D, 'NOP', is highlighted with a green box. The instruction at address 0101E, 'NOP', is highlighted with a green box. The instruction at address 0101F, 'NOP', is highlighted with a green box. The instruction at address 01020, '...', is highlighted with a green box. The instruction at address 01010, 'MOV AL, 0Fh', is highlighted with a blue box. The instruction at address 01011, 'MOV [00001h], AL', is highlighted with a green box. The instruction at address 01012, 'MOV BL, 0Ah', is highlighted with a green box. The instruction at address 01013, 'ADD AL, BL', is highlighted with a green box. The instruction at address 01014, 'MOV [00000h], AL', is highlighted with a green box. The instruction at address 01015, 'INT 016h', is highlighted with a green box. The instruction at address 01016, 'NOP', is highlighted with a green box. The instruction at address 01017, 'NOP', is highlighted with a green box. The instruction at address 01018, 'NOP', is highlighted with a green box. The instruction at address 01019, 'NOP', is highlighted with a green box. The instruction at address 0101A, 'NOP', is highlighted with a green box. The instruction at address 0101B, 'NOP', is highlighted with a green box. The instruction at address 0101C, 'NOP', is highlighted with a green box. The instruction at address 0101D, 'NOP', is highlighted with a green box. The instruction at address 0101E, 'NOP', is highlighted with a green box. The instruction at address 0101F, 'NOP', is highlighted with a green box. The instruction at address 01020, '...', is highlighted with a green box.

registers	H	L
AX	00	00
BX	00	00
CX	00	00
DX	00	00
CS	0100	
IP	0010	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

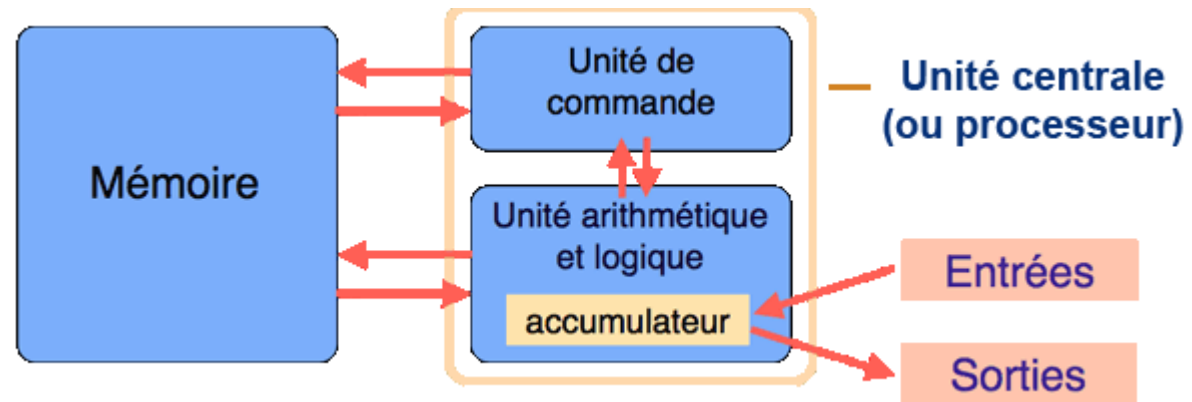
flags	
CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

address	hex	ascii
01000	01 001	
01001	02 002	
01002	00 000	NULL
01003	00 000	NULL
01004	00 000	NULL
01005	00 000	NULL
01006	00 000	NULL
01007	00 000	NULL
01008	00 000	NULL
01009	00 000	NULL
0100A	00 000	NULL
0100B	00 000	NULL
0100C	00 000	NULL
0100D	00 000	NULL
0100E	00 000	NULL
0100F	00 000	NULL
01010	B0 176	
01011	0F 015	
01012	A2 162	
01013	01 001	
01014	00 000	NULL
01015	B3 179	
01016	0A 010	NEWL
01017	02 002	
01018	C3 195	
01019	A2 162	
0101A	00 000	NULL
0101B	00 000	NULL
0101C	CD 205	
0101D	16 022	
0101E	90 144	
0101F	90 144	
01020	90 144	

address	instruction
01000	ADD [BP + SI], AX
01001	ADD [BX + SI], AL
01002	ADD [BX + SI], AL
01003	ADD [BX + SI], AL
01004	ADD [BX + SI], AL
01005	ADD [BX + SI], AL
01006	ADD [BX + SI], AL
01007	ADD [BX + SI], AL
01008	ADD [BX + SI], AL
01009	ADD [BX + SI], AL
0100A	ADD [BX + SI], AL
0100B	ADD [BX + SI], AL
0100C	ADD [BX + SI], AL
0100D	ADD [BX + SI], AL
0100E	ADD [BX + SI], AL
0100F	ADD [BX + SI], AL
01010	MOV AL, 0Fh
01011	MOV [00001h], AL
01012	MOV BL, 0Ah
01013	ADD AL, BL
01014	MOV [00000h], AL
01015	INT 016h
01016	NOP
01017	NOP
01018	NOP
01019	NOP
0101A	NOP
0101B	NOP
0101C	NOP
0101D	NOP
0101E	NOP
0101F	NOP
01020	...

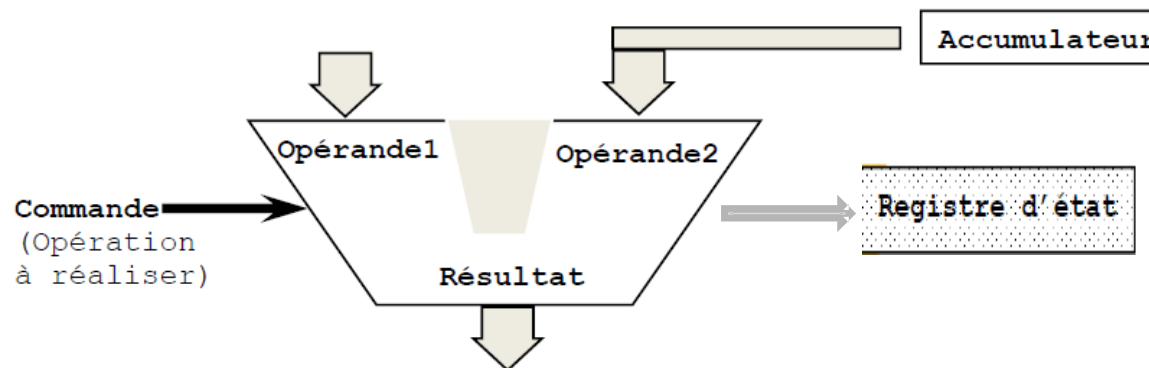
Revenons à l'archi de Von Neumann

- Unité de commande (UC), qui organise le séquençement des instructions
- Compteur ordinal contient l'adresse de l'instruction en cours d'exécution,
 - soit incrémenté
 - modifié lors des sauts
- Unité arithmétique UAL, exécute les instructions arithmétiques et logiques
- Accumulateur (ou registres) : mémorise les résultats intermédiaires des calculs avant E/S
- Mémoire peut contenir les programmes et les données
 - Programme : suite d'instructions codées selon un format conventionnel (jeu d'instruction)
 - Données : valeurs encodées selon un format conventionnel (entier, chaîne, flottants ...)



Unité Arithmétique et logique

- ▶ Reçoit de L'UAL les **opérations** à effectuer et les **opérandes** associées.
- ▶ Composée de circuits logiques qui réalisent :
 - des opérations **arithmétiques classique pour un CPU** : + - * ...**logique**: or and not
 - Des **opérations plus complexes** : pour un FPU/GPU : inverse, racine carré, produit scalaire, vectoriel



UAL : Exemple d'opération

- ▶ Opération de **comparaison** du contenu d'un registre avec 0

CMP AL, 0



Instruction de
comparaison en
assembleur 8086



Si AL==0
Alors ZF=1
Sinon ZF=0
Finsi

- Flag ZF du registre d'état
- ▶ Une UAL fait un nombre limité d'opérations. Fonctions manquantes à programmer

Registres d'état

Indicateur (généralement composé de 8 bits) dont l'état dépend du résultat de la dernière opération effectuée par l'UAL.

Indicateurs d'état ou flags.

► ZF (Zero Flag)

- mis à 1 lorsque le résultat de la dernière opération est zéro.
- sinon, ZF est positionné à 0.

► CF (Carry Flag)

- Indicateur de report (retenue). Il est positionné par les instructions ADD, SUB et CMP (entiers naturels).
- $CF = 1$ s'il y a une retenue

Registres d'état

▶ SF (Sign Flag)

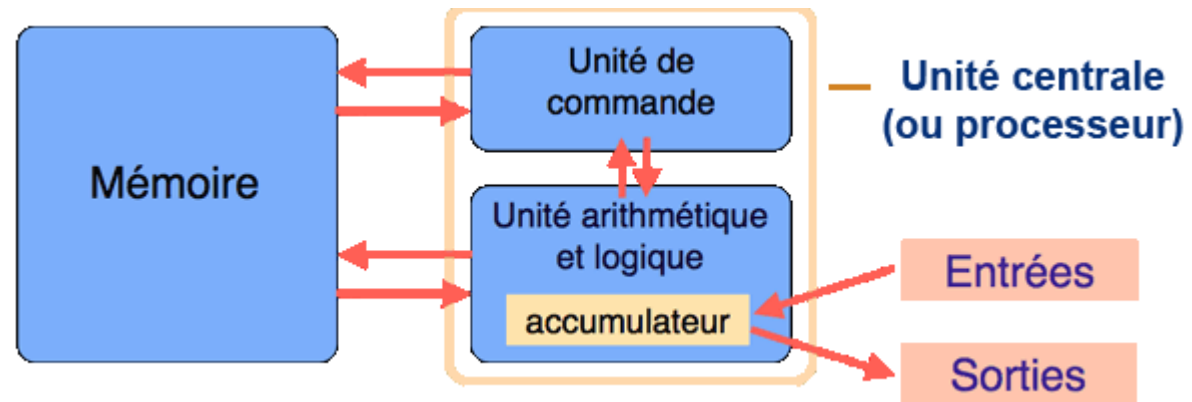
- Positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1; sinon SF=0. Utile pour la manipulation des entiers relatifs, le bit de poids fort donne le signe du résultat.

▶ OF (Overflow Flag) Indicateur de débordement

- OF=1 si le résultat d'une addition ou soustraction donne un nombre qui n'est pas codable en relatif dans l'accumulateur (par exemple si l'addition de 2 nombres positifs donne un codage négatif).

Revenons à l'archi de Von Neumann

- Unité de commande (UC), qui organise le séquençement des instructions
- Compteur ordinal contient l'adresse de l'instruction en cours d'exécution,
 - soit incrémenté
 - modifié lors des sauts
- Unité arithmétique UAL, exécute les instructions arithmétiques et logiques
- **Accumulateurs** (ou registres) : mémorise les résultats intermédiaires des calculs avant E/S
- Mémoire peut contenir les programmes et les données
 - Programme : suite d'instructions codées selon un format conventionnel (jeu d'instruction)
 - Données : valeurs encodées selon un format conventionnel (entier, chaîne, flottants ...)



Accumulateurs

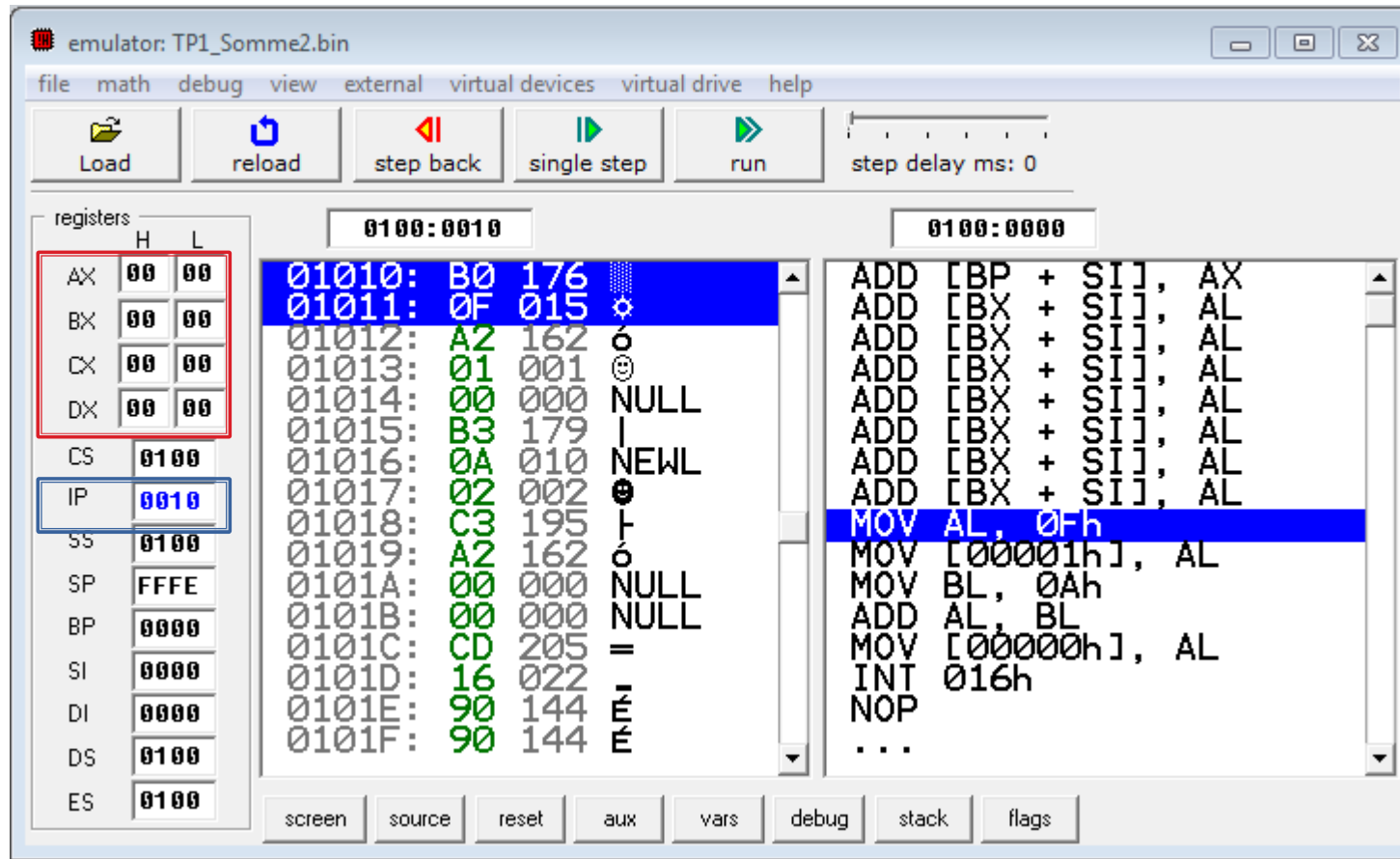
- ▶ **Mémoire interne** proche de l'UAL (accès rapide)
- ▶ 4 registres de données,
 - Décomposés en deux parties : une partie « haute » et une partie « basse » de 8 bits chacune, ce qui permet au microprocesseur de manipuler des données sur 8 ou 16 bits :
 - AX (décomposable en AH et AL) sert d'accumulateur et est principalement utilisé lors d'opérations arithmétiques et logiques ;
 - BX est la plupart du temps utilisé comme opérande dans les calculs ;
 - CX est utilisé comme compteur dans les structures itératives ;
 - DX, tout comme AX, est utilisé pour les calculs arithmétiques et notamment dans la division et la multiplication. Il intervient également dans les opérations d'entrées/sorties.

Accumulateurs

Visualisation sous l'émulateur 8086

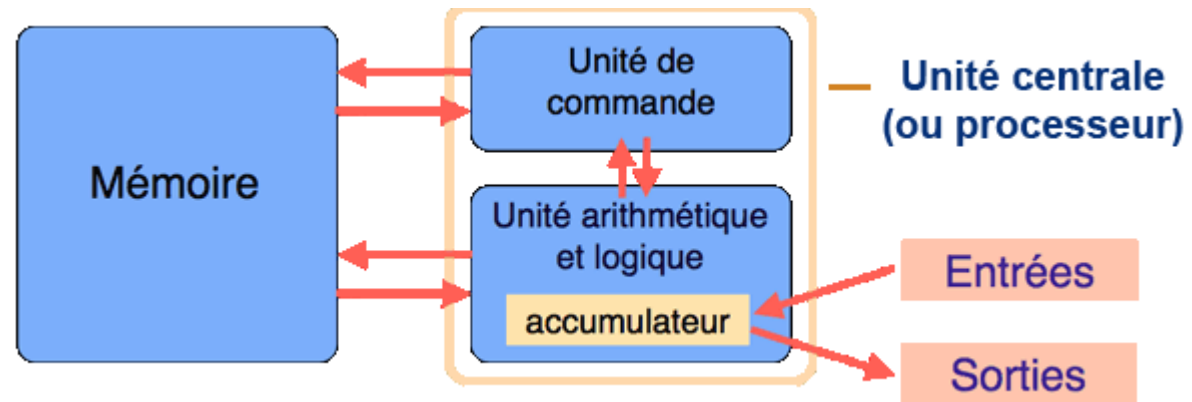
Accu-
mulateurs

Compteur
Ordinal



Revenons à l'archi de Von Neumann

- Unité de commande (UC), qui organise le séquençement des instructions
- Compteur ordinal contient l'adresse de l'instruction en cours d'exécution,
 - soit incrémenté
 - modifié lors des sauts
- Unité arithmétique UAL, exécute les instructions arithmétiques et logiques
- Accumulateurs (ou registres) : mémorise les résultats intermédiaires des calculs avant E/S
- Mémoire peut contenir les programmes et les données
 - Programme : suite d'instructions codées selon un format conventionnel (jeu d'instruction)
 - Données : valeurs encodées selon un format conventionnel (entier, chaîne, flottants ...)



Mémoire

- ▶ Une mémoire peut être représentée comme une armoire de rangement constituée de différents tiroirs.
 - un tiroir représente alors une **case mémoire** qui peut contenir un seul élément **un octet**.
 - Une **adresse** identifie la case mémoire et chaque donnée est accessible grâce à son adresse
- ▶ On peut accéder simultanément à plusieurs tiroirs via des **bus d'adresses** et de **données**
- ▶ Exemple : avec un bus de données 64 bits on peut accéder à 4 tiroirs (4 octets) simultanément

Mémoire

- ▶ **Stocke** des informations utilisées par le processeur
- ▶ **Juxtaposition** de cellules (la plus petite quantité de mémoire adressable).
 - Chaque cellule est numérotée (adresse)
 - Généralement les mémoires sont adressables par octets
- ▶ **Taille** d'une cellule
 - octet : 8 bits (byte)
 - word : 16 bits,
 - Double word : 32 bits,
 - Quad word 64 bits : en fonction de machines (16 ou 32 bits)
- ▶ **Notation** :
 - Si M désigne une cellule mémoire
 - [M] représente son contenu

Mémoire : décomposition en segments

- ▶ **Principe** : la numérotation sur 16 bits a ses limites (2^{16} cases adressables : 65535 octets (un octet = 8 bits))
- ▶ **Notion de segments** : consiste à regrouper les cases mémoires par zone plus grande (segment) de 65535 octets
- ▶ Exemple sur l'émulateur 8086:

Segment 0000

De 0000:0000
à 0000:FFFF

Segment 0001

De 0001:0000
à 0001:FFFF

Segment 0003

De 0002:0000
à 0002:FFFF

0000:0000	70	01	00	F4	00	01	00	F4	00	01	00	F4	00	01	00	F4
0000:0010	80	01	00	F4	00	01	00	F4	00	01	00	F4	00	01	00	F4
0000:0020	00	01	00	F4	00	01	00	F4	00	01	00	F4	00	01	00	F4
0000:FFD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000:FFE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000:FFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0001:0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0001:0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0001:FFB0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0001:FFC0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0001:FFD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0001:FFE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0001:FFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0002:0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0002:0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Quelle est l'adresse de l'octet cerclé de rouge ? 0000:002A

Mémoire programme, données

Données

Registres
Segments

Programme

The screenshot displays a debugger window with three main sections:

- Registers:** A list of registers (AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, DS, ES) with their current values. The CS register is highlighted with a blue box, and the DS register is highlighted with a red box.
- Memory:** A window titled "Random Access Memory" showing a memory dump. The address 0100:0000 is highlighted with a red box. The dump shows hexadecimal values and their corresponding ASCII representations.
- Program Code:** A window showing the disassembled code. The instruction at address 01010: is highlighted with a blue box.

The memory dump shows the following data:

Address	Hex	ASCII
0100:0000	01 02 00 00 00 00 00 00	00-00 00 00 0
0100:0010	B0 0F A2 01 00 B3 0A 02-C3	A2 00 00 0
0100:0020	90 90 90 90 90 90 90 90	90-90 90 90 9
0100:0030	90 90 F4 00 00 00 00 00	00-00 00 00 0
0100:0040	00 00 00 00 00 00 00 00	00-00 00 00 0
0100:0050	00 00 00 00 00 00 00 00	00-00 00 00 0
0100:0060	00 00 00 00 00 00 00 00	00-00 00 00 0
0100:0070	00 00 00 00 00 00 00 00	00-00 00 00 0
0100:0080	00 00 00 00 00 00 00 00	00-00 00 00 0

The program code window shows the following instructions:

Address	Instruction
01010:	B0 176
01011:	0F 015
01012:	A2 162
01013:	01 001
01014:	000 NULL