



Chapitre 3 : Notions sur les instructions d'un ordinateur

Partie 1 : 'Instruction machine'

COURS ARCHITECTURE DES ORDINATEURS

PRÉSENTÉ PAR : BAQACH ADIL

Plan cours 5

Partie 1 du

Chapitre 3 :

***Notions sur les
instructions d'un
ordinateur***

-
- I. Langages de programmation**
 - II. Instruction machine**
 - III. Principe de compilation et d'assemblage**

Machines multi-niveaux

5. Langages haut niveau

Compilation

4. Langage d'assemblage

Assembleur

3. Système d'exploitation

Appels système

2. Jeu d'instructions propre à chaque machine

Microprogrammes : micro-instructions binaires

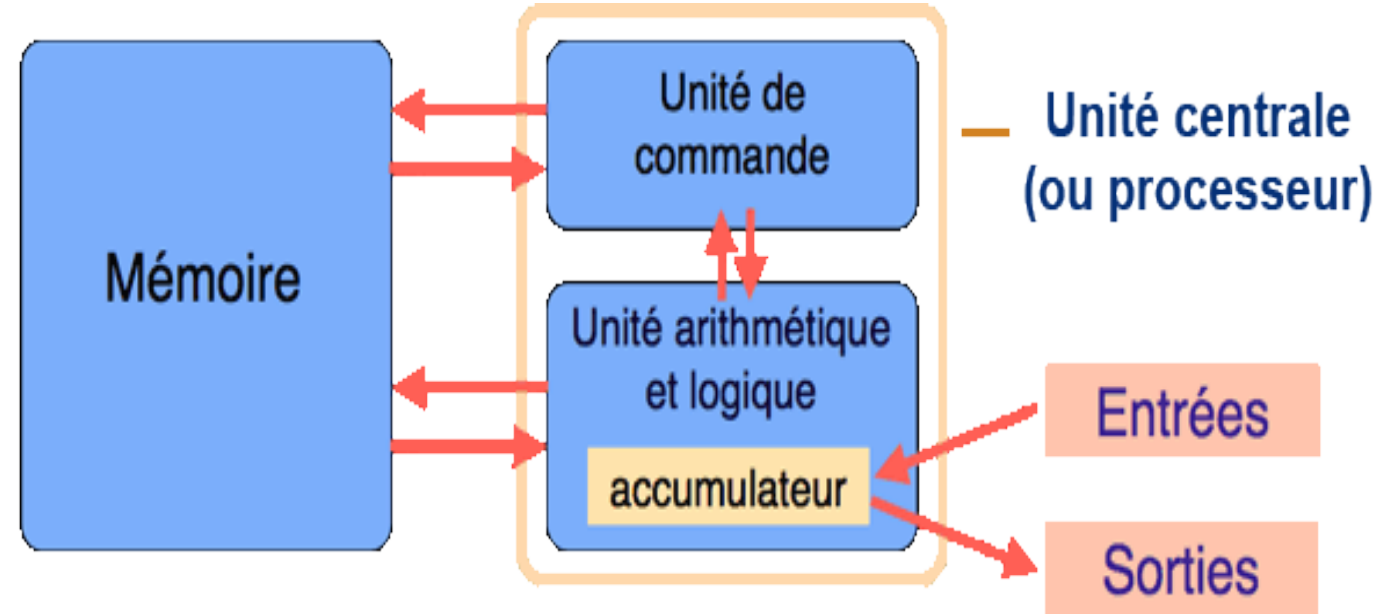
1. Micro-architecture (UAL, opérations, registres, ...)

Assemblage physique des portes logiques

0. Circuits logiques

Comment un programme s'exécute-t-il dans l'ordinateur ?

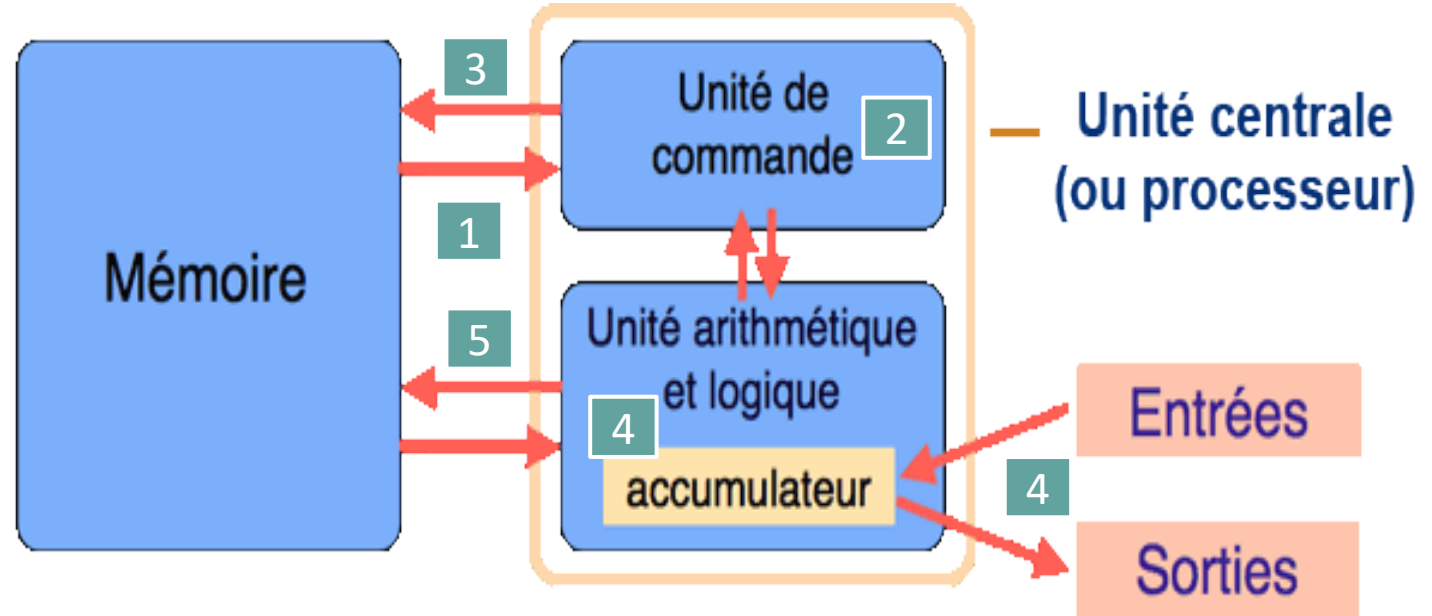
- la machine est construite autour d'un processeur, véritable tour de contrôle interne, d'une mémoire stockant les données et le programme et d'un dispositif d'entrées/sorties nécessaire pour l'échange avec l'extérieur.



Comment un programme s'exécute-t-il dans l'ordinateur ?

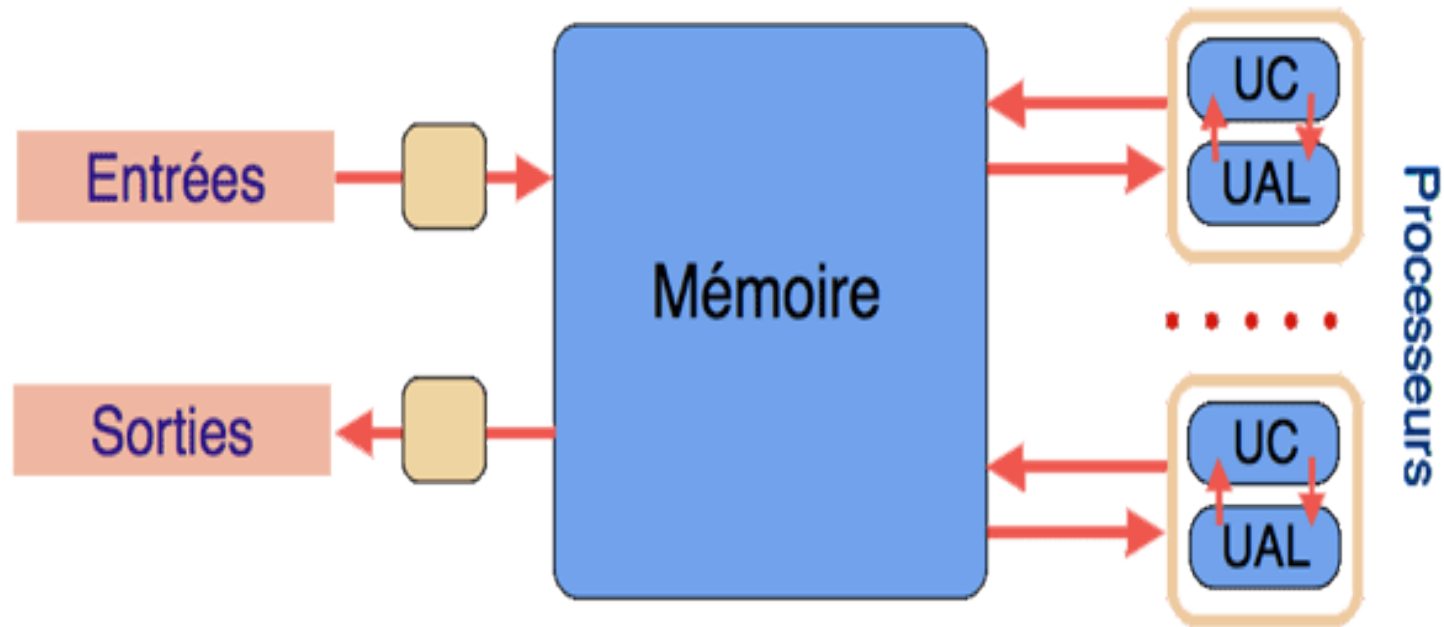
Le déroulement d'un programme au sein de l'ordinateur est le suivant:

1. L'UCC extrait une instruction de la mémoire
2. Analyse l'instruction
3. Recherche dans la mémoire les données concernées par l'instruction
4. Déclenche l'opération adéquate sur l'UAL ou l'E/S
5. Range au besoin le résultat dans la mémoire



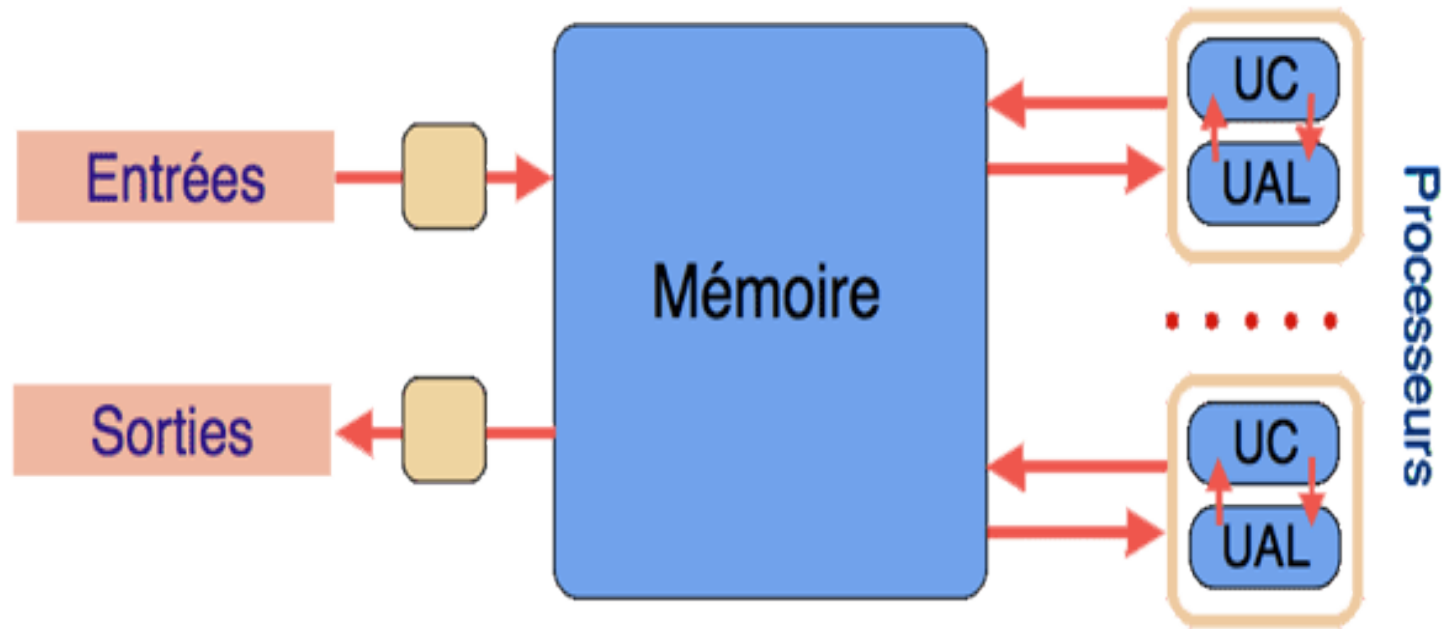
Qu'en est il de l'architecture aujourd'hui?

- Plus de soixante ans après son invention, le modèle d'architecture de von Neumann régit toujours l'architecture des ordinateurs.
- Par rapport au schéma initial, on peut noter deux évolutions :



Qu'en est il de l'architecture aujourd'hui?

- Les **entrées-sorties**, initialement commandées par l'unité centrale, sont depuis le début des années 1960 sous le contrôle de processeurs autonomes (canaux d'entrée-sortie et mécanismes assimilés).
- Les ordinateurs comportent maintenant des **processeurs multiples**, qu'il s'agisse d'unités séparées ou de « cœurs » multiples à l'intérieur d'une même puce.



1- Langages de programmation

LA **PROGRAMMATION** EST DONC L'ACTIVITÉ QUI CONSISTE À TRADUIRE, PAR UN PROGRAMME, UN **ALGORITHME** DANS UN **LANGAGE ASSIMILABLE PAR L'ORDINATEUR.**

Cette activité de programmation peut s'effectuer à différents niveaux :

- la programmation de bas niveau en **langage machine**,
- la programmation de bas niveau en **langage d'assemblage**,
- la programmation de haut niveau à l'aide d'un **langage de haut niveau** ou langage évolué.

1. Langage machine

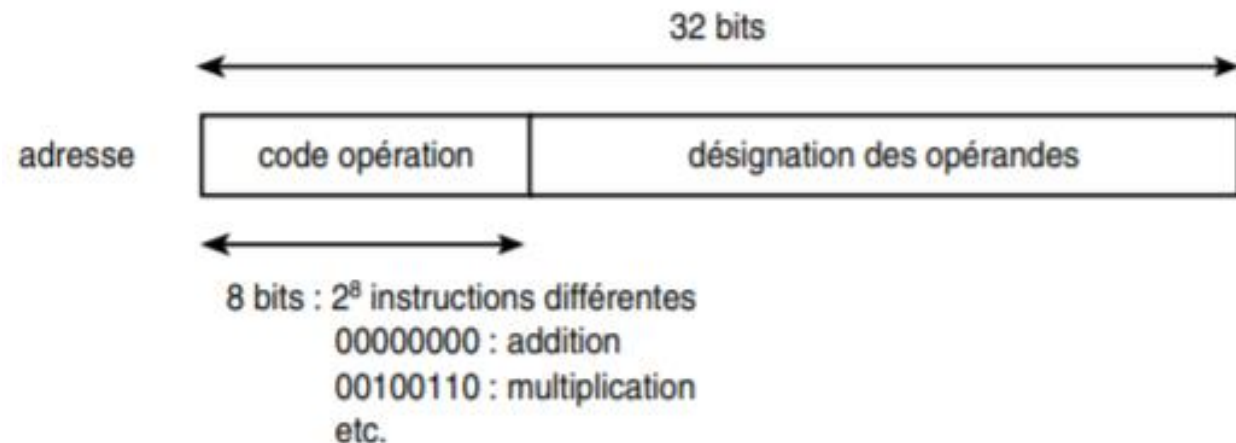
- C'est le seul langage exécutable directement par le microprocesseur.
- Ce langage est difficile à maîtriser puisque chaque instruction est codée par une séquence propre de bits (Binary Digit).
- Afin de faciliter la tâche du programmeur, on a créé différents langages plus ou moins évolués.

Langage machine

```
01001011 11001010
01011001 10011011
01001011 11001010
01011101 10011011
```

1. Langage machine

- Une **instruction machine** est une chaîne binaire composée essentiellement de deux parties : le **code opération** et les **opérandes**.
- Ces opérandes sont soit des mots mémoires, soit des registres du processeur ou encore des valeurs immédiates.
- Chaque instruction est par ailleurs repérée par une adresse qui mémorise la position de l'instruction dans le programme.



2. Langage assembleur

- C'est le langage le plus proche du langage machine.
- Il est composé par des instructions en général assez rudimentaires que l'on appelle des **mnémoniques**.
- Ce sont essentiellement **des opérations de transfert de données** entre les registres et l'extérieur du microprocesseur (mémoire ou périphérique), ou **des opérations arithmétiques ou logiques**.
- Chaque instruction représente un code machine différent et chaque microprocesseur peut posséder un assembleur différent.

Assembleur

```
mov dx,offset text
mov ah,09h
int 21h
mov ah,4ch
...
```

2. Langage assembleur

Une **instruction du langage d'assemblage** est composée de champs séparés par un ou plusieurs espaces. On identifie :

- Un champ **étiquette** : non obligatoire, qui correspond à l'adresse de l'instruction machine.
- Un champ **code opération** : qui correspond à la chaîne binaire code opération de l'instruction machine ;
- Un champ **opérandes** : pouvant effectivement comporter plusieurs opérandes séparés par des virgules qui correspondent aux registres, mots mémoires ou valeurs immédiates apparaissant dans les instructions machine.

étiquette	code opération	désignation des opérandes
-----------	----------------	---------------------------

l'instruction en langage d'assemblage

étiquette	code opération	opérandes
boucle :	ADD	Rg2 R0, R1

correspond à l'instruction machine

adresse	code opération	opérandes
01110110	00000000	111 0000 0001

3. Langage haut niveau (ou évolué)

- La difficulté de mise en œuvre de ce type de langage (machine et assembleur) et leur forte dépendance avec la machine a nécessité la conception de **langages de haut niveau**, plus adaptés à l'homme, et aux applications qu'il cherchait à développer.
- Le langage de haut niveau est le niveau de programmation le plus utilisé aujourd'hui.
- C'est un niveau de programmation indépendant de la structure physique de la machine et de l'architecture du processeur qui permet l'expression d'algorithmes sous une forme plus facile à apprendre et à dominer.



3. Langage haut niveau (ou évolué)

Deux familles de langages importants et courants sont :

- **Le langage procédural** : l'écriture d'un programme est basée sur les notions de procédures et de fonctions, qui représentent les traitements à appliquer aux données du problème, de manière à aboutir à la solution du problème initial. **Les langages C et Pascal sont deux exemples de langages procéduraux**
- **Le langage objet** : l'écriture d'un programme est basée sur la notion d'objets, qui représentent les différentes entités entrant en jeu dans la résolution du problème. À chacun de ces objets sont attachées des méthodes, qui lorsqu'elles sont activées, modifient l'état des objets. **Les langages Java et Eiffel sont deux exemples de langages objets.**

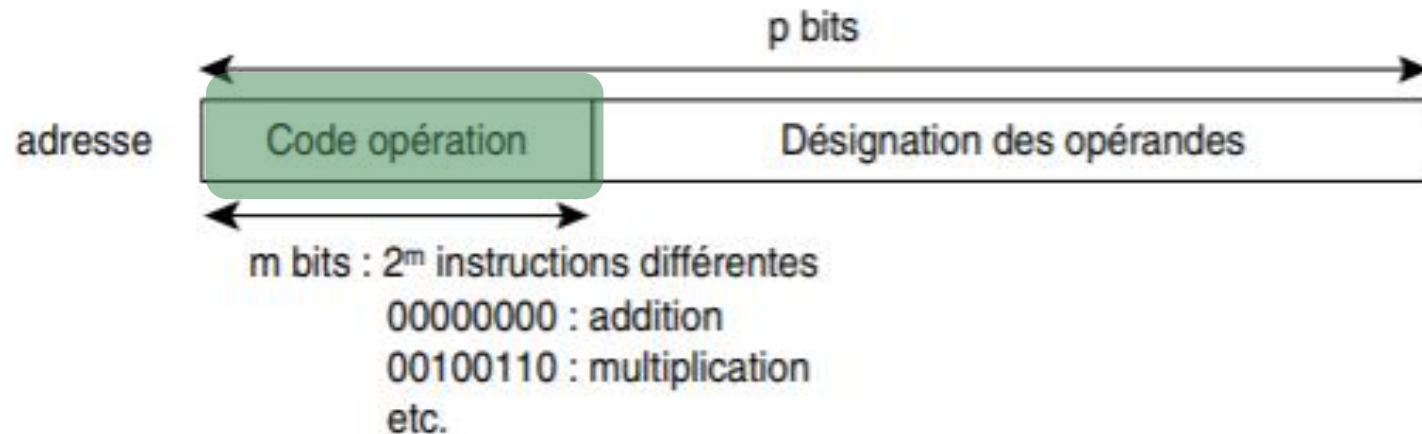


2- Instruction machine

UNE **INSTRUCTION** (EN LANGAGE MACHINE OU ASSEMBLEUR) DÉSIGNE
UN **ORDRE DONNÉ AU PROCESSEUR** ET QUI PERMET À CELUI-CI DE
RÉALISER UN **TRAITEMENT ÉLÉMENTAIRE**.

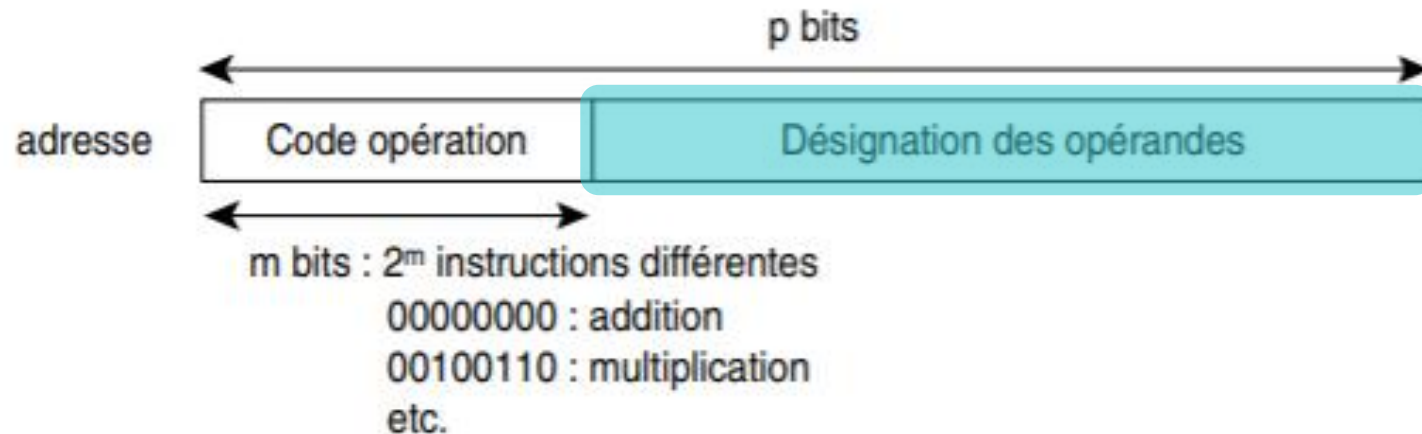
L'instruction machine est une chaîne binaire de **p** bits composée principalement de deux parties :

- Le champ **code opération** composé de **m** bits :
- ❖ Il indique au processeur le type de traitement à réaliser (addition, lecture d'une case mémoire, etc.).
- ❖ Un code opération de **m** bits permet de définir **2^m** opérations différentes pour la machine.
- ❖ Le nombre d'opérations différentes autorisées pour une machine définit **le jeu d'instructions** de la machine.

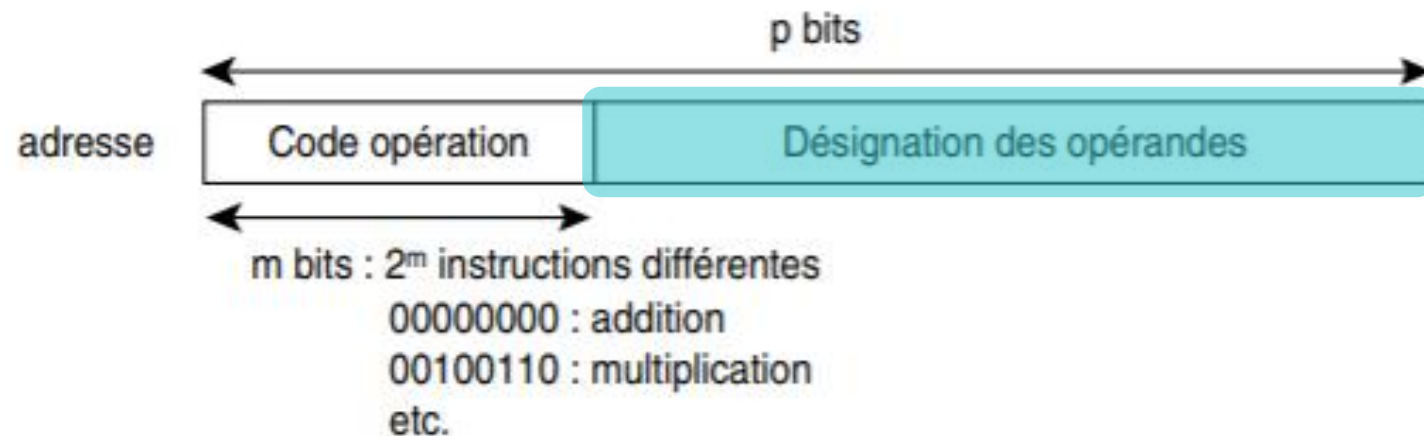


L'instruction machine est une chaîne binaire de **p** bits composée principalement de deux parties :

- Le champ **opérandes** composé de **p – m** bits :
- ❖ Il permet d'indiquer la nature des données sur lesquelles l'opération désignée par le code opération doit être effectuée.
- ❖ La façon de désigner un opérande dans une instruction peut prendre différentes formes : on parle alors de **mode d'adressage** des opérandes.



- La taille d'une instruction dépend du type de l'instruction et du type de l'opérande.
- Les instructions et leurs opérandes sont stockés dans la mémoire



Remarque :

- Chaque architecture de la machine physique correspond à une forme symbolique du langage machine associé au processeur.
- Pour éviter d'avoir affaire au langage machine difficilement manipulable par l'homme on utilise un langage symbolique équivalent appelé langage assembleur.
- Les codes opérations d'une instruction machine sont représentés par des abréviations appelés **mnémonique** indiquant les opérations

1. Classification des machines par le nombre d'opérandes

Il est parfois fait référence à une machine par le **nombre de champs opérandes** contenus dans ses instructions. Il existe des **machines à 4, 3, 2, 1 ou 0 adresse (s)**.

1. Classification des machines par le nombre d'opérandes

Machine à 4 adresses

C.Opé

Opé 1

Opé 2

Résultat

Inst. suivante

A1 : adresse du 1^{er} opérande ou la donnée elle-même

A2 : adresse du 2^{ème} opérande ou la donnée elle-même

A3 : adresse où doit être rangé le résultat

A4 : adresse de l'instruction suivante

Effet : $A3 \leftarrow (A1) + (A2)$

Inst Suiv = A4

Exemple : ADD 19,13,100,110 Effet : $100 \leftarrow (19) + (13)$ Inst Suiv = 110

Dans ce type de machine la taille de l'instruction est très grande.

1. Classification des machines par le nombre d'opérandes

Machine à 3 adresses

Le champ A4 n'existe pas. L'adresse de l'instruction suivante étant indiquée par le compteur ordinal (CO).

C. Ope	Opé 1	Opé 2	Résultat
--------	-------	-------	----------

Effet : $A3 \leftarrow (A1) + (A2)$

CO \leftarrow **CO** + 1

Exemple : ADD 19,13,100

Effet : $100 \leftarrow (19) + (13)$

Dans ce type de machine la taille de l'instruction est grande

1. Classification des machines par le nombre d'opérandes

Machine à 2 adresses

Les champs A4 et A3 n'existent pas. Le résultat est rangé dans le mot dont l'adresse est contenue dans le champ A2.

C. Opé	Opé 1	Opé 2
--------	-------	-------

Effet : $A2 \leftarrow (A1) + (A2)$

CO \leftarrow **CO** + 1

Exemple : ADD 19,13

Effet : $13 \leftarrow (19) + (13)$

l'adresse de résultat est implicitement l'adresse du deuxième opérande.

1. Classification des machines par le nombre d'opérandes

Machine à 1 adresse

A1 : désigne l'emplacement du 1^{er} opérande. Le 2^{ème} opérande se trouve dans un registre, l'opération terminée, le résultat est rangé dans ce registre. Sur certaines machines, ce registre est appelé **accumulateur**.

C. Opé	Opérande
--------	----------

Effet : $ACC \leftarrow (ACC) + (A1)$

Exemple : ADD 19 Effet : $ACC \leftarrow 100 + (19)$ / Si ACC contient la valeur 100

Ce type de machine est le plus utilisé

1. Classification des machines par le nombre d'opérandes

Machine à 0 adresse « machine à pile »

Appelée machine à pile, ces machines n'ont pas de vrai registre, toutes les opérations effectuent un transfert entre la mémoire et une pile.

Exemple : ADD

Effet : si contient val0 et val1 \rightarrow pile={val0+val1}

2. Modes d'adressage des opérandes

- Par **mode d'adressage**, on désigne le chemin que doit emprunter l'unité centrale (processeur) pour accéder à l'opérande.
- Il désigne comment le champ adresse de l'instruction est utilisé pour déterminer l'opérande.

2. Modes d'adressage des opérandes

Remarque :

- **Le mode d'adressage**, se trouve indiqué au sein de l'instruction dans le code opération ou dans un champ séparé réservé à cet effet, appelé conditions d'adressage.
- **L'adresse effective**, correspond à l'adresse finale envoyée dans le RAM après des transformations du contenu de la partie adresse de l'instruction.

2. Modes d'adressage des opérandes

Adressage Immédiat

L'opérande contenue dans un champ de l'instruction ne désigne pas l'adresse (emplacement dans un registre ou une mémoire) où se trouve la valeur. Il désigne la valeur elle-même.

Exemple :

LOAD nbr ou **LOAD #nbr**

$ACC \leftarrow nbr$

ADD 150 Cette commande va avoir l'effet suivant : **$ACC \leftarrow ACC + 150$**

Si le registre accumulateur contient la valeur 200 alors
après l'exécution son contenu sera égale à 350

2. Modes d'adressage des opérandes

Adressage Direct

L'adresse de l'opérande est donnée dans l'instruction sous forme d'adresse mémoire.

Exemple :

LOAD adr **$\text{!ACC} \leftarrow M[\text{adr}]$** **Adresse effective : adr**

ADD 150 Cette commande va avoir l'effet suivant : **ACC ← ACC + (150)**

ACC ← ACC + 30

Si le registre accumulateur contient la valeur 200
alors après l'exécution son contenu sera égale à 230

2. Modes d'adressage des opérandes

Adressage Indirect

C'est l'emplacement où se trouve l'adresse qui est désigné et non l'adresse elle-même.

Exemple :

LOAD [adr] ou LOAD @adr **!ACC ← M[M[adr]]** Adresse effective : M[adr]

ADD 150 Cette commande va avoir l'effet suivant : **ACC ← ACC + ((150))**

ACC ← ACC + (255)

ACC ← ACC + 40

Si le registre accumulateur contient la valeur 200 alors après l'exécution son contenu sera égale à 240

2. Modes d'adressage des opérandes

Il y a aussi :

-Adressage registre : Exemple : LOAD R1 **!ACC ← R1**

-Adressage registre indirect : Exemple :

LOAD (R1)	$!ACC \leftarrow M[R1]$	si R1 contient une adresse mémoire
	$!ACC \leftarrow R1$	si R1 contient un n° de registre

-Adressage relatif : Adresse effective = (ADR base) + déplacement (valeur du registre).

Example : LOAD 100(R1) !ACC ← M[100+R1]

-Adressage indexée : Adresse effective = (ADR base) + déplacement (valeur du registre d'index).

Exemple : LOAD (adr+RI) !ACC ← M[adr+RI]

-Adresse auto-incrémenté et auto-décrémenté : (ex accès à un tableau)

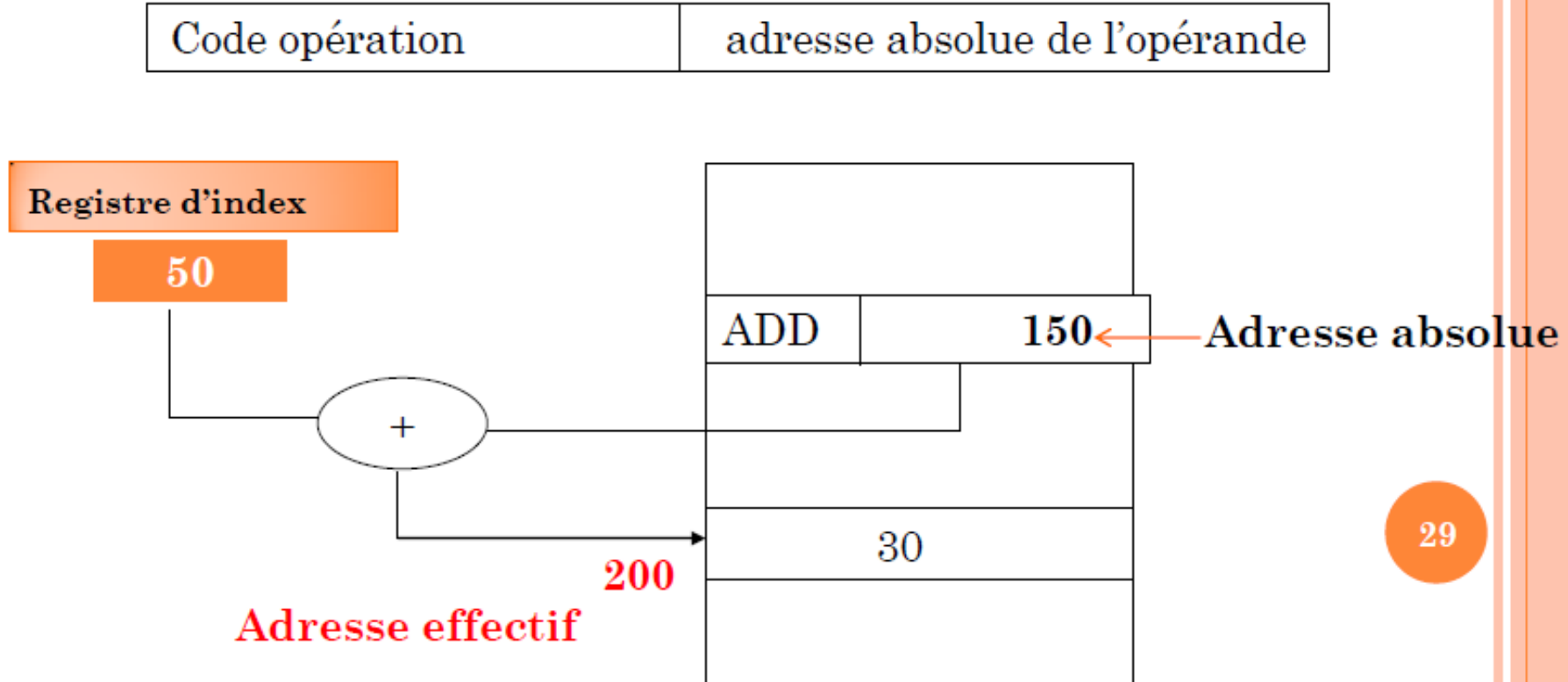
Example : ADD R1, (R2)+ !R1 ← R1 + M[R2]

$!R1 \leftarrow R1 + d$ d : facteur de déplacement

2. Modes d'adressage des opérandes

Adressage indexée : Adresse effective = (ADR base) + déplacement (valeur du registre d'index).

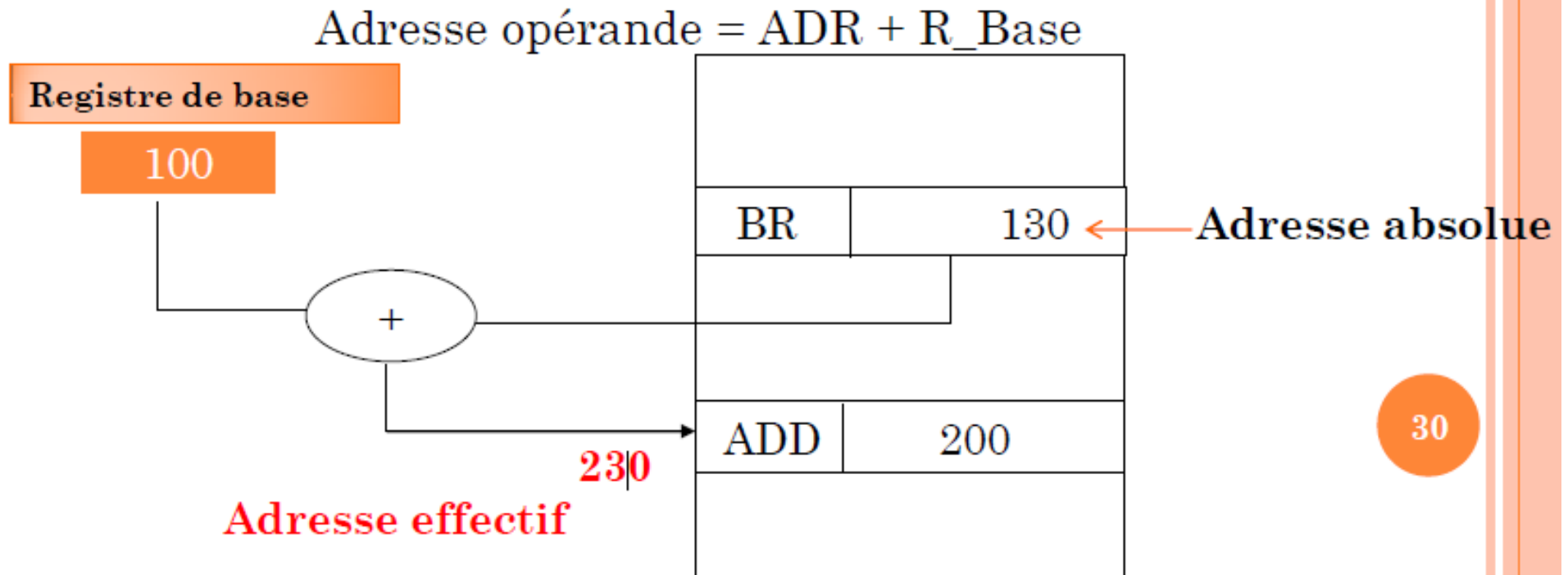
$$\text{Adresse opérande} = \text{ADR} + \text{R_Index}$$



2. Modes d'adressage des opérandes

Adressage relatif : Adresse effective = (ADR base) + déplacement (valeur du registre).

Ce mode d'adressage est utilisée pour les instructions de branchement.



Soit l'instruction
suivante :
 $A \leftarrow 3 + 5$.

Réécrire cette
instruction en une
suite d'instructions de
format à deux (02)
adresses, de format à
une (01) adresse et
format à zéro (0)
adresse.

Machine à 0 adresse (Machine à pile)	Machine à 1 adresse (Machine à accumulateur)	Machine à 2 adresses (Machine a registres)	
PUSH X $\text{Top (pile)} \leftarrow X$	LOAD X $\text{Acc} \leftarrow X$	MOVE R_i , X $R_i \leftarrow X$	
POP X $X \leftarrow \text{Top (pile)}$	STORE X $X \leftarrow \text{Acc}$	STORE R_i , X $X \leftarrow R_i$	
ADD $\text{POP } t_1; \text{POP } t_2$ $\text{PUSH } t_1 + t_2$	ADD X $\text{Acc} \leftarrow \text{Acc} + x$	ADD R_i , R_j $R_j \leftarrow R_i + R_j$	ADD X, R_i $R_i \leftarrow X + R_i$
PUSH 3 PUSH 5 ADD POP A	LOAD 3 ADD 5 STORE A	MOVE R1, 3 MOVE R2, 5 ADD R1, R2 STORE R2, A	MOVE R1, 3 ADD 5, R1 STORE R1, A

3. Différents types d'instructions

a. les instructions arithmétiques et logiques : Ces deux groupes d'instructions mettent en jeu les circuits de l'UAL.

Add, Sub, Mul, Div et *And, Or, Not, Xor*, etc....

b. Instructions pour déplacer l'information : pour charger la mémoire, sauvegarder en mémoire, effectuer des transferts de registres à la mémoire, registre à registre, de mémoire à mémoire, etc...

Load, Move, Store

c. Instructions de contrôle du programme (sauts et branchement) **et instructions** qui vérifient certaines **conditions d'état** (comparaison).

JMP, JMPO

3. Différents types d'instructions

d. les instructions d'entrées-sorties : ce sont les instructions qui permettent au processeur de lire une donnée depuis un périphérique (par exemple le clavier) ou d'écrire une donnée vers un périphérique (par exemple l'imprimante).

e. les instructions particulières permettent par exemple d'arrêter le processeur (**HALT**) ou encore de masquer/démasquer les interruptions (**DI/EI**).

Instruction et synonymes	Signification (en anglais)	Condition nécessaire
JZ	Jump if Zero flag is set	Indicateur Zéro à 1
JNZ	Jump if No Zero flag is set	Indicateur Zéro à 0
JE	Jump if both operand were Equals	ou quand les deux opérandes A et B sont égaux ($A-B = 0$)
JNE	Jump if both operand were Not Equals	ou quand les deux opérandes A et B ne sont pas égaux ($A-B \neq 0$)
JS	Jump if Sign flag is set	Indicateur de signe à 1 (résultat négatif)
JNS	Jump if No Sign flag is set	Indicateur de signe à 0 (résultat positif)

Instruction et synonymes	Signification (en anglais)	Condition nécessaire
JO	Jump if Overflow flag is set	Indicateur de débordement à 1
JNO	Jump if No Overflow flag is set	Indicateur de débordement à 0
JNAE	Jump if Not Above or Equal	Indicateur de retenue à 1 ($A < B$, c'est à dire $A-B < 0$, retenue)
JAE	Jump if Above or Equal	Indicateur de retenue à 0 ($A \geq B$, c'est à dire $A-B \geq 0$, pas de retenue)
JNA	Jump if Not Above	Indicateur de retenue à 1 ou indicateur Zéro à 1 ($A < B$ ou $A = B$, c'est à dire $A-B \leq 0$)
JA	Jump if Above	Indicateur de retenue à 0 et indicateur Zéro à 0 ($A \geq B$ et $A \neq B$, c'est à dire $A > B$, $A-B > 0$)

Adressage indexée : Exercise

Un tableau d'entiers est stocké en mémoire à partir de l'adresse de base 2000. Le tableau contient 5 éléments : [10, 20, 30, 40, 50].

Écrire un programme en assembleur qui utilise un registre d'index pour parcourir le tableau, additionner chaque élément dans le registre accumulateur (ACC) et afficher la somme totale.

2. Modes d'adressage des opérandes

Adressage indexée : Exercise

MOV 2000, R1 ; R1 contient l'adresse de base du tableau

MOV 0, RI ; Initialiser le registre d'index RI à 0

MOV 5, R2 ; R2 contient la taille du tableau (5 éléments)

MOV 0, ACC ; Initialiser l'accumulateur à 0

LOOP:

ADD (R1 + RI) ; Ajouter l'élément à l'adresse (R1 + RI) à l'accumulateur

ADD 1, RI ; Incrémenter l'index RI pour accéder à l'élément suivant

CMP RI, R2 ; Comparer l'index avec la taille du tableau

JNAE LOOP ; Si RI n'est pas supérieur ou égal à R2, revenir à LOOP

Adressage relatif : Exercise

Un tableau d'entiers est stocké en mémoire à partir de l'adresse de base 2000. Le tableau contient 5 éléments : [10, 20, 30, 40, 50].

Écrire un programme en assembleur qui utilise un registre d'index pour parcourir le tableau, additionner chaque élément dans le registre accumulateur (ACC) et afficher la somme totale.

Adressage relatif : Exercise

MOV 4000, R1 ; R1 contient l'adresse de base du tableau
MOV 0, R2 ; Initialiser R2 à 0 pour le déplacement (index)
MOV 5, R3 ; R3 contient la taille du tableau (5 éléments)
MOV 0, ACC ; Initialiser l'accumulateur ACC à 0

LOOP:

ADD M[R1 + R2] ; Ajouter l'élément à l'adresse (R1 + R2) à ACC
ADD 1, R2 ; Incréments R2 pour passer à l'élément suivant
CMP R2, R3 ; Comparer R2 avec la taille du tableau
JAE END ; Si R2 >= R3, sortir de la boucle

JMP LOOP ; Sinon, revenir au début de la boucle

END: