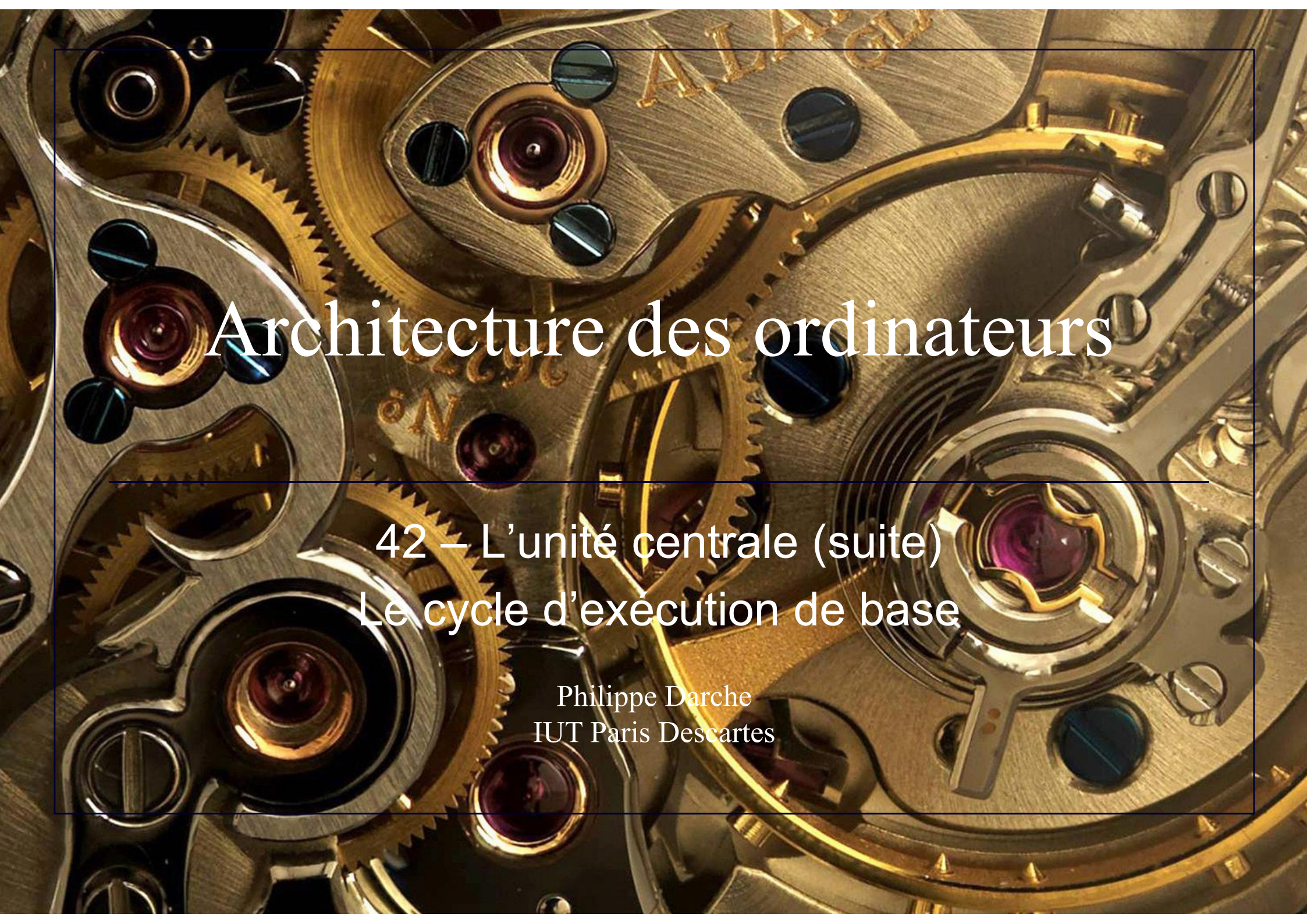
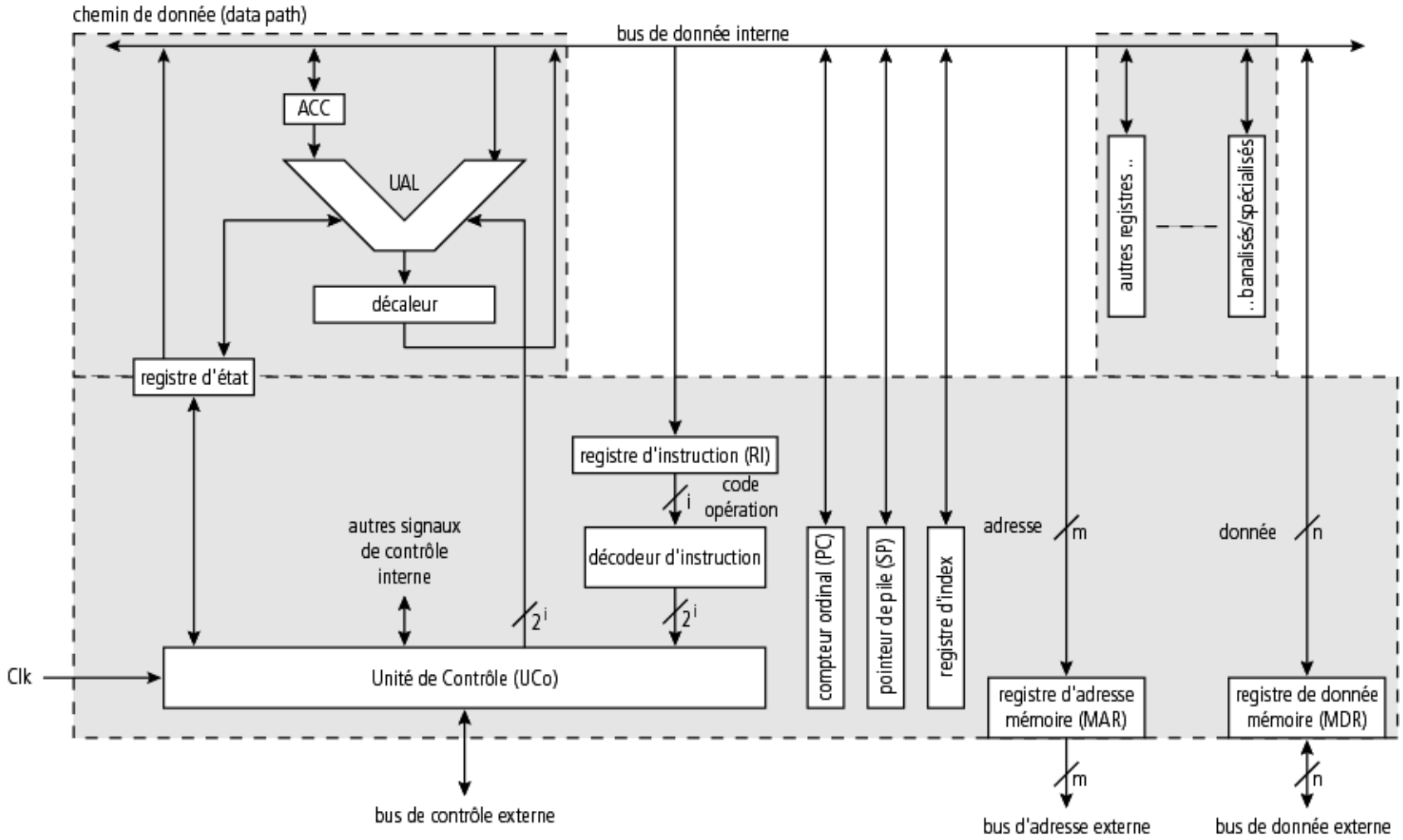


Architecture des ordinateurs

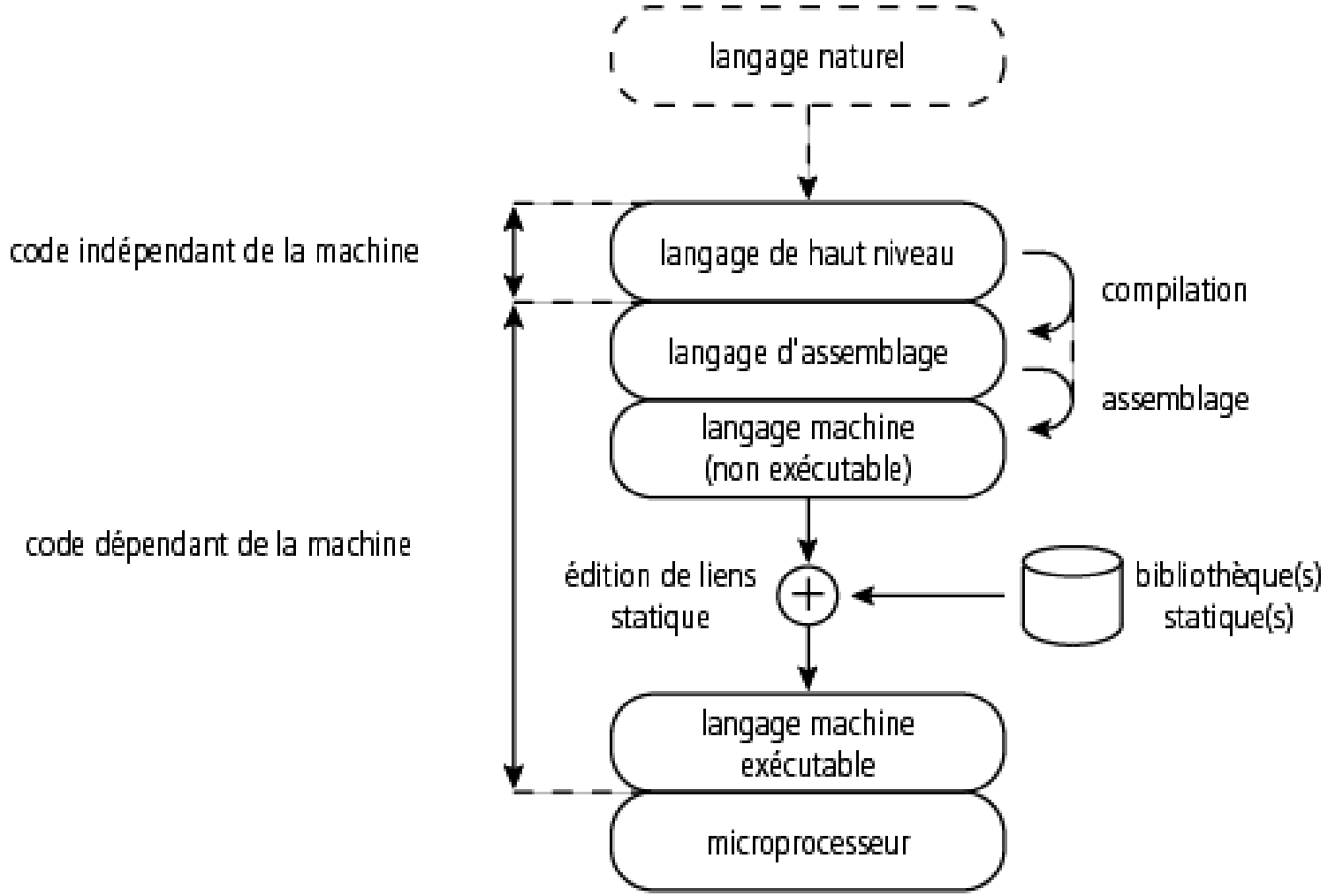
42 – L'unité centrale (suite)
Le cycle d'exécution de base

Philippe Darche
IUT Paris Descartes

Une micro-architecture simple à bus de donnée unique (rappel)



Vue en couches conceptuelles



Architecture de jeu d'instructions

- En anglais, *Instruction Set Architecture* (ISA)
- Couche d'abstraction
  - indique comment programmer le CPU (au-dessus)
  - indique les fonctionnalités à implémenter (en-dessous)
- Définit l'interface entre le matériel et le logiciel
  - spécification fonctionnelle
    - jeu d'instructions et l'encodage des instructions
      - langage d'assemblage
    - détails des registres manipulables et de l'organisation mémoire
- = Vue logique visible par le programmeur de bas niveau
- ≠ Microarchitecture (= détails internes)

Microarchitecture

- Implémentation d'une ISA
- Utilisation d'une UCo et d'une UT
- Notion de chemin (*path*)
  - ensemble des composants ou sous-ensembles traversés par un type de d'information
    - chemin de données (*data path*)
    - chemin d'adresse (*address path*)

Instructions

- Jeu d'instructions

  = {instructions exécutables par le processeur}

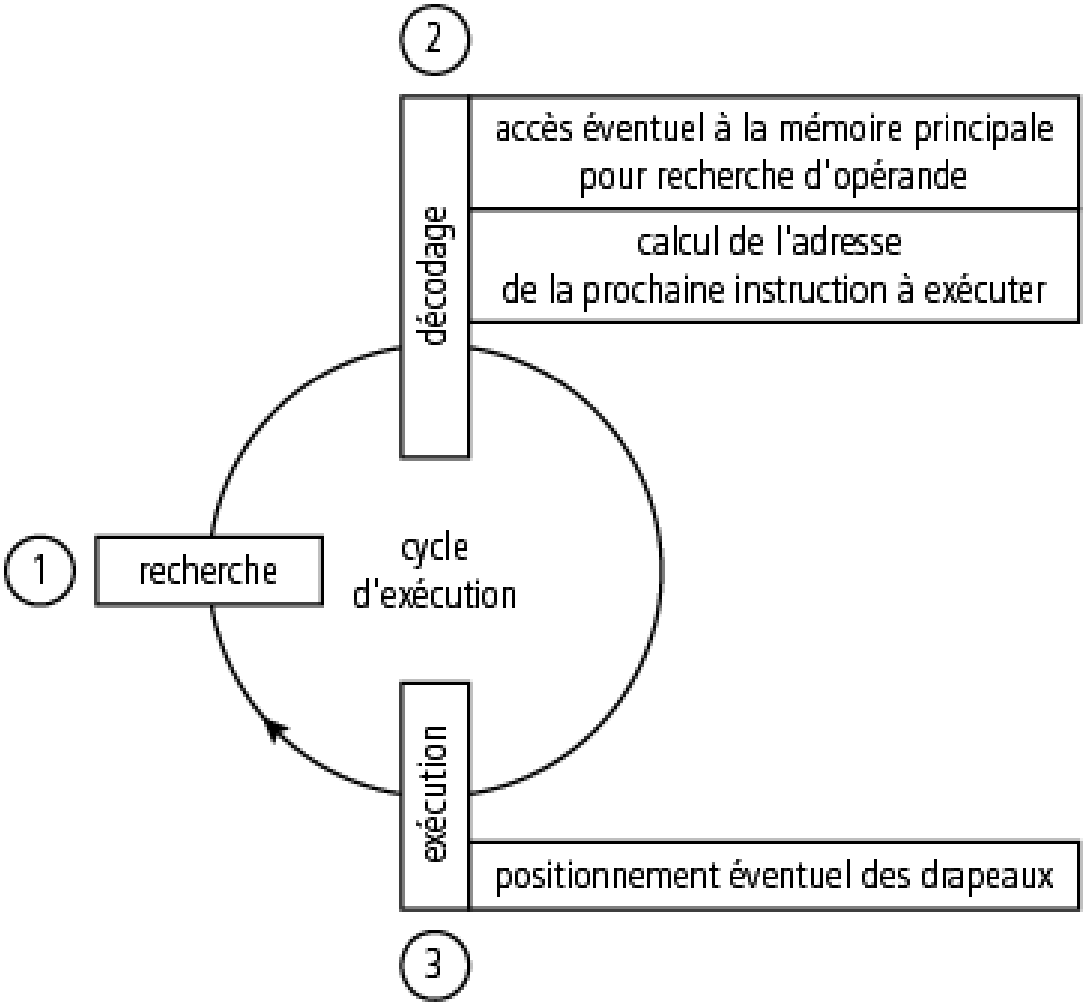
- Familles d'instructions
  - transferts de données
  - arithmétiques
  - logiques
  - transferts de contrôle
  - E/S

Quelques exemples chez Intel

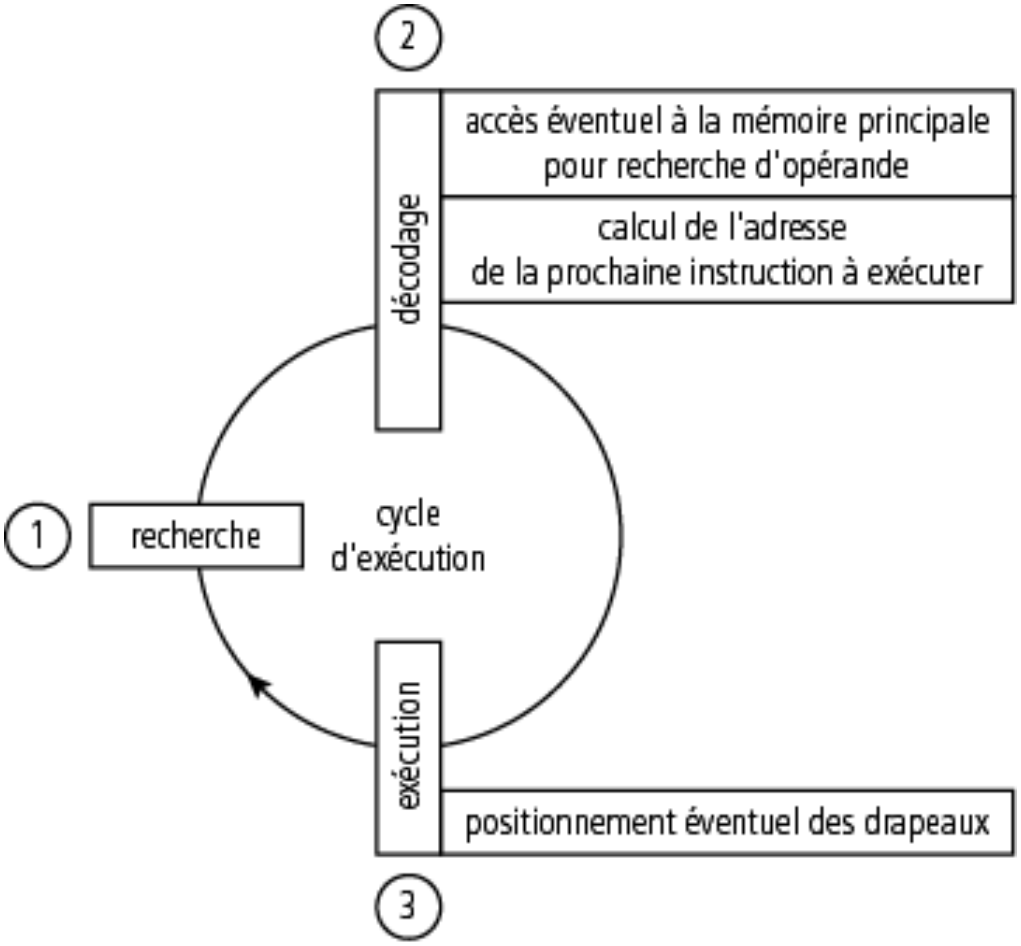
- Transfert de données : mov
- Arithmétiques
  - add, inc, sub, dec, mul, div
  - shl, shr, sal, sar, rol, ror, rcl, rcr
- Logiques : not, and, or, xor
- Transfert de contrôle
  - jmp
  - je, jz, jne, jze, js, jse, jl, jle, jnle, jae, jnb, ja, jae, etc.
  - jo, jno
  - loop
- E/S : in, out

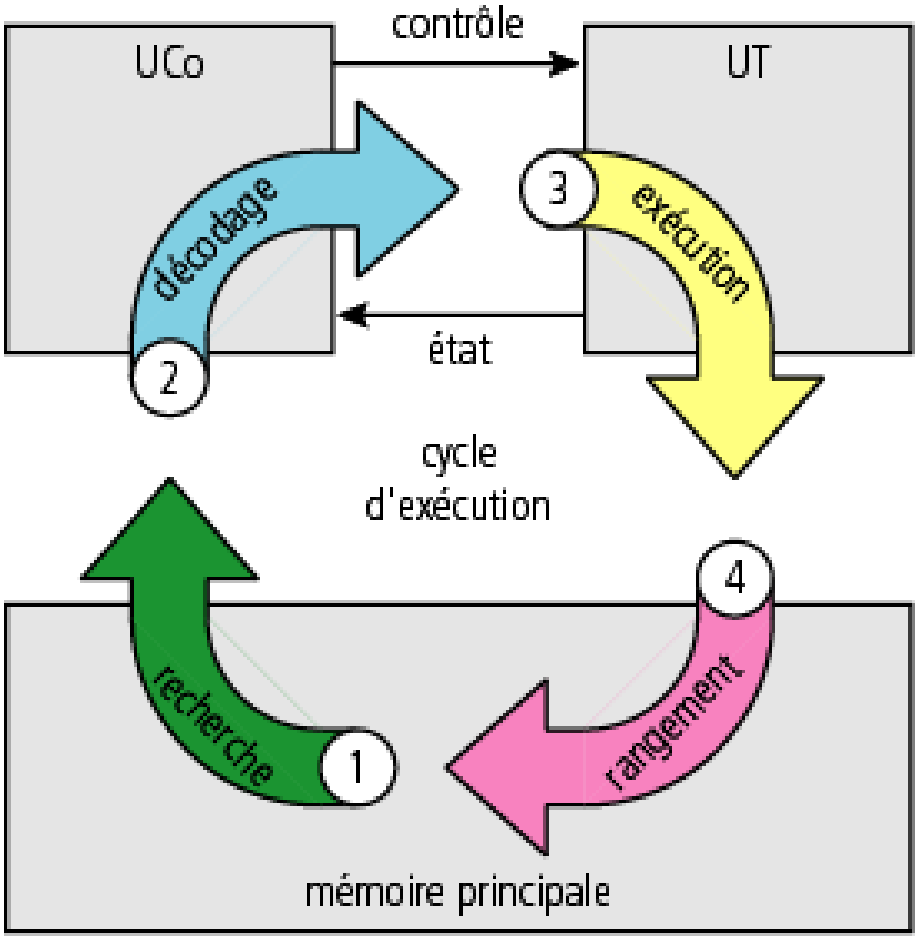
Cycle d'exécution de base

- Décomposé en :
  - une étape de recherche
    - *fetch cycle*
  - une étape de décodage
    - *decode cycle*
  - une étape d'exécution
    - *execute cycle*
  - ⇒ Boucle infinie



Relation avec la mémoire principale





Le cycle d'exécution de base

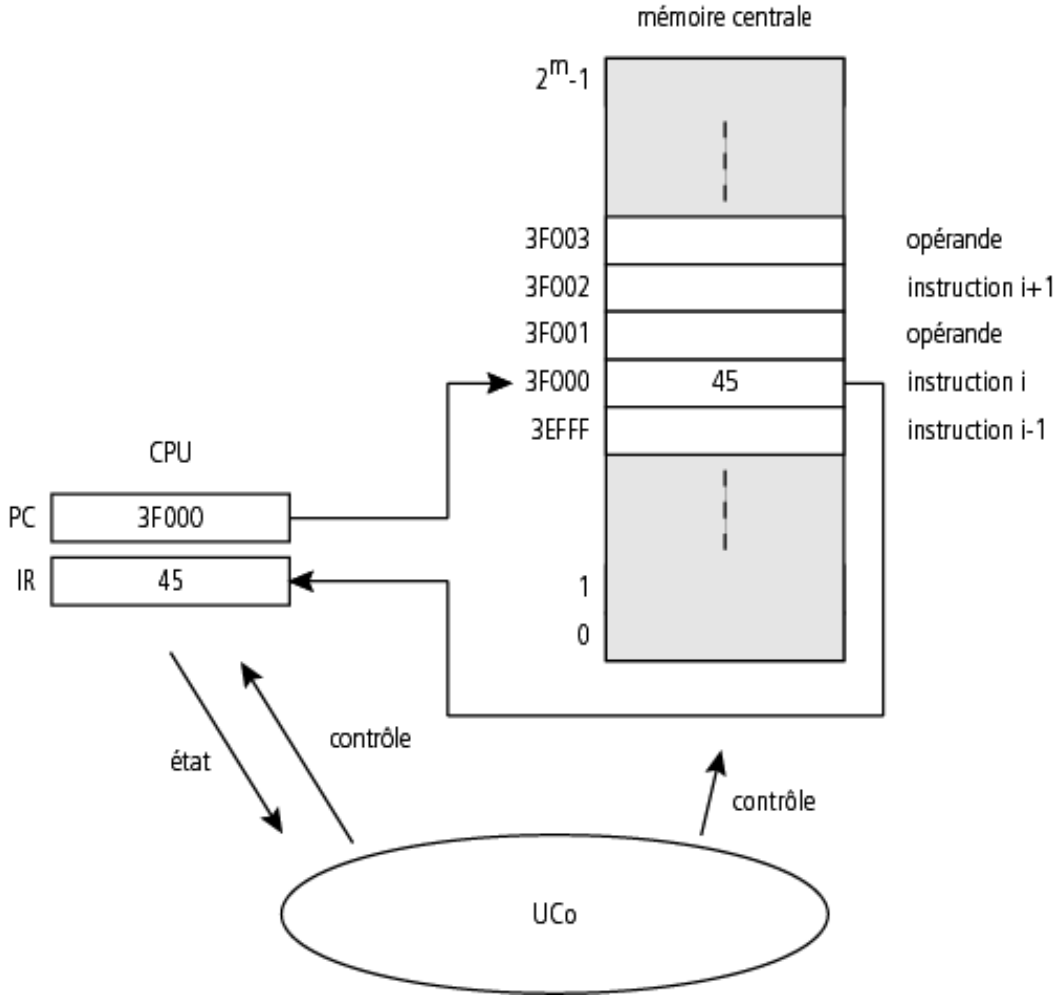


Schéma d'exécution de base

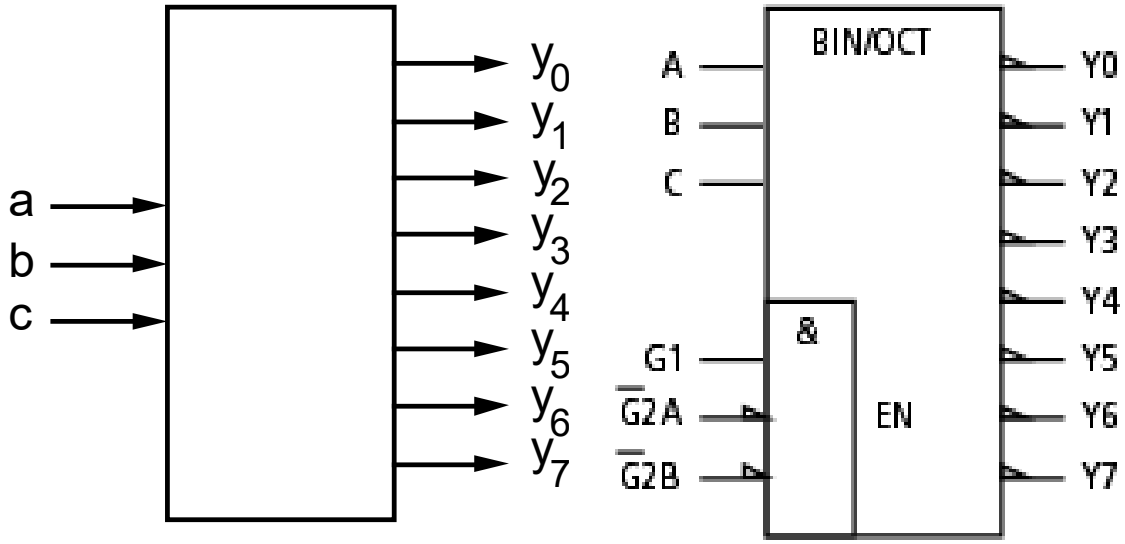
- Etape de recherche
  - recherche du code opération en mémoire centrale
  - stockage dans un registre inaccessible
    - le registre d'instruction (IR pour *Instruction Register*)
- Etape de décodage du code machine
  - décodage du champ fonction de l'instruction
  - mise à jour du compteur ordinal (CO ou IP chez Intel)
  - recherche éventuelle d'un opérande en mémoire centrale
  - stockage dans un accumulateur ou traitement direct

Décodage d'une instruction

- Chaque code instruction préalablement stocké dans IR est décodé par le décodeur d'instructions
  - rappel sur le décodeur binaire
  - une sortie par instruction

Le décodeur binaire naturel

- n entrées, p = 2^n sorties
  - d'où l'appellation décodeur n vers p
- Exemple : le décodeur 3 vers 8



Poids										
2^2	2^1	2^0								
c	b	a	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Le décodeur d'instruction

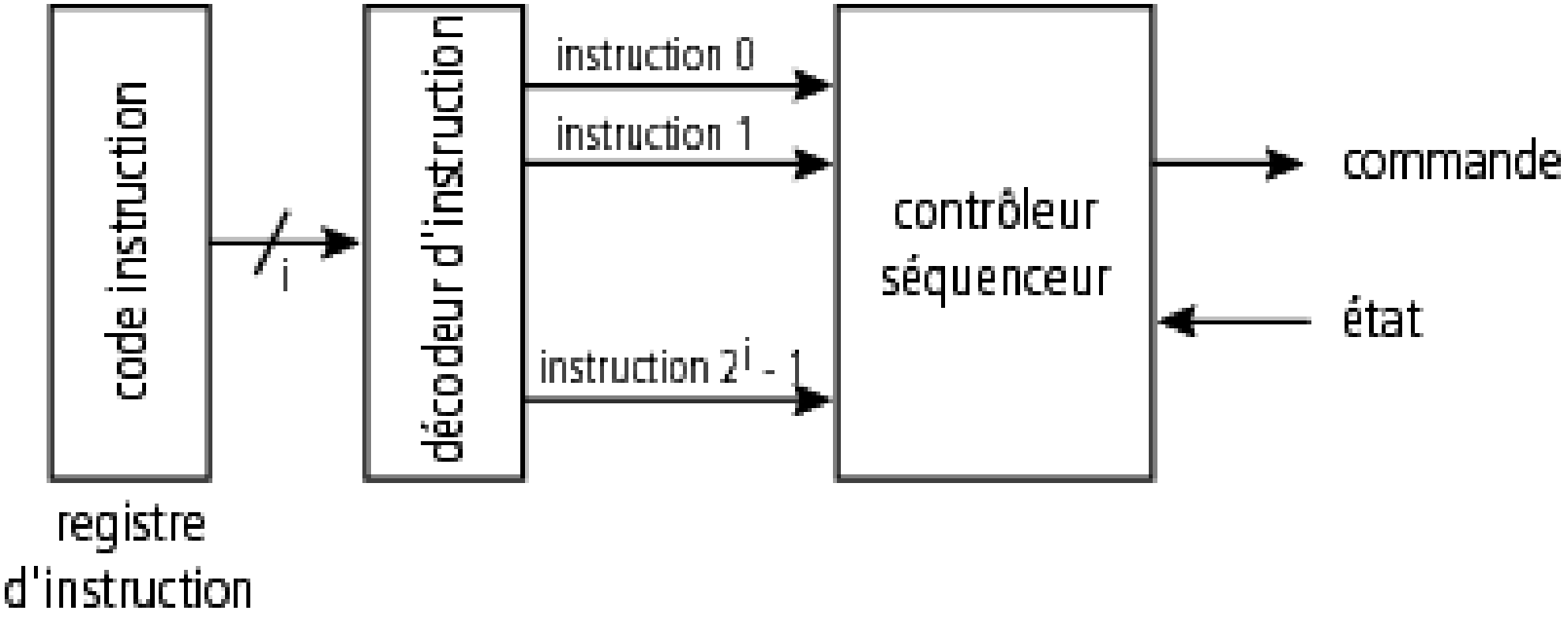
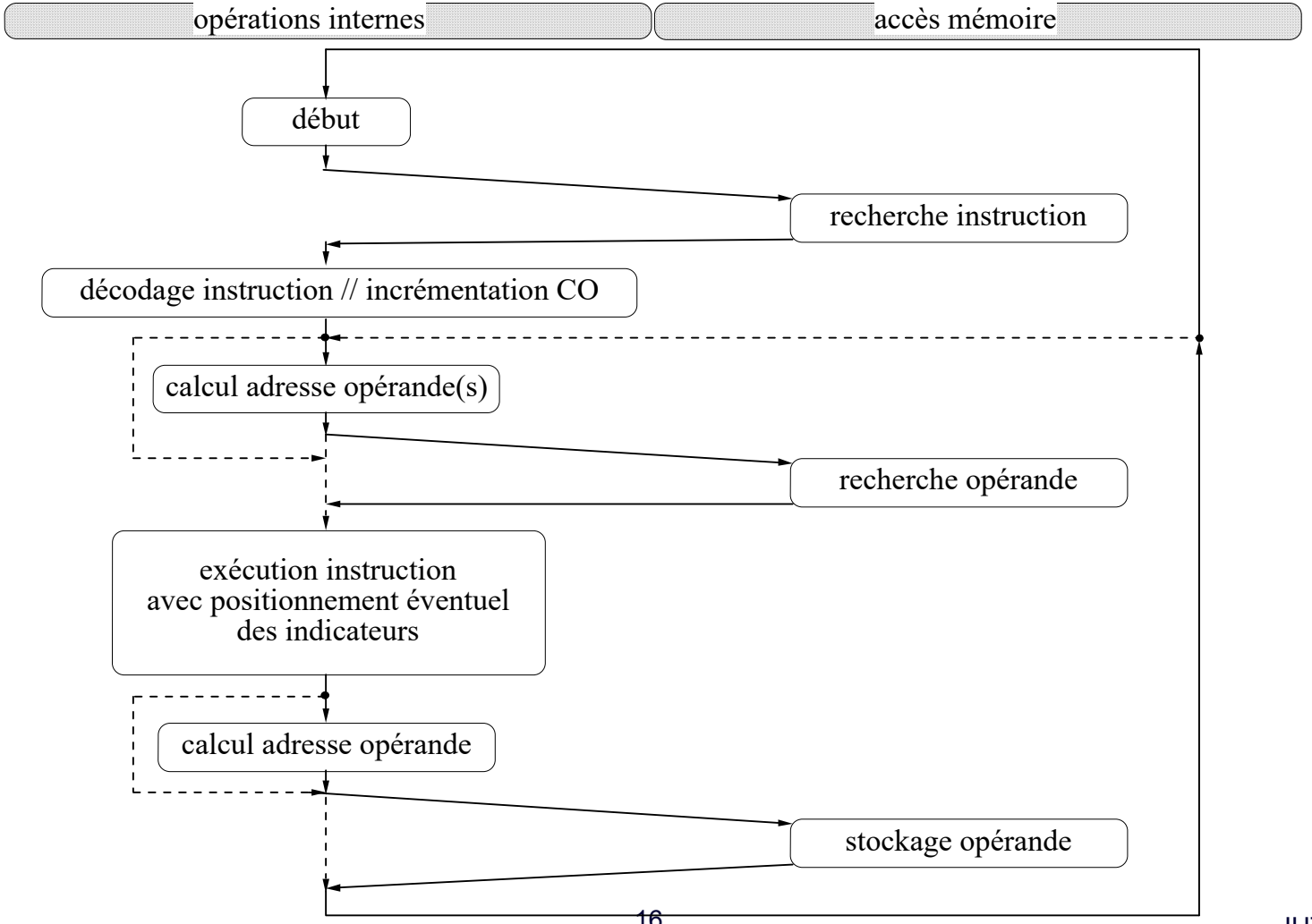


Schéma d'exécution de base

- Etape d'exécution
  - stockage du résultat dans un registre appelé accumulateur ou rangement en mémoire
  - positionnement éventuel des indicateurs binaires du registre d’état

Cycle d'exécution détaillé avec accès mémoire pour l'opérande



Instructions en langage d'assemblage

- Utilisation d'un mnémonique plus lisible plutôt qu'un code opération
  - abréviation d'instruction
- Exemples :
  - nop; ne rien faire
  - mov ax,[y]; ax ← (y)
  - sub ax,[y]; ax ← (ax) - (y)
- Outil de traduction : l'assembleur

Exemple d'une instruction et son codage

- NOP (*No Operation*)
  - pas d'opérande
  - fonction : ne rien faire ! Mais :
    - utilisation pour retarder le CPU (boucle d'attente logicielle)
    - réservation de place mémoire dans l'image binaire d'un programme pour une éventuelle rustine (*patch*)
    - test du bus d'adresse

Exemple d'une instruction et son codage

- ADD (*ADDition*)
  - syntaxe assembleur : add destination,source
  - deux opérandes (opérateur binaires)
  - fonction : addition de deux entiers
  - opération
    - destination ← (destination) + (source)

La règle d'or de l'accès aux opérandes sources

- le CPU ne peut faire, au plus, qu'un accès à la mémoire principale pour les opérandes sources (*i.e.* nécessaires au calcul)
  - s'il faut plus d'un accès (deux opérandes par exemple), utilisation des registres internes (AX, etc.)
- Illustration des combinaisons possibles : instruction ADD

  (tableau du bas p. 3-64 de 8086_documentation_1_scan.pdf)

ADD — ADDITION

Operation (DEST) ← (LSRC) + (RSRC)

Flags Affected AF, CF, OF, PF, SF, ZF

Description **ADD** *destination, source*

The sum of the two operands, which may be bytes or words, replaces the destination operand. Both operands may be signed or unsigned binary numbers (see AAA and DAA). ADD updates AF, CF, OF, PF, SF and ZF.

Encoding

Memory or Register Operand with Register Operand

0 0 0 0 0 0 d w	mod reg r/m

if d = 1 then LSRC = REG, RSRC = EA, DEST = REG
else LSRC = EA, RSRC = REG, DEST = EA

Immediate Operand to Memory or Register Operand

1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01

LSRC = EA, RSRC = data, DEST = EA

Immediate Operand to Accumulator

0 0 0 0 0 1 0 w	data	data if w = 1

if w = 0 then LSRC = AL, RSRC = data, DEST = AL
else LSRC = AX, RSRC = data, DEST = AX

ADD Operands	Clocks	Transfers	Bytes	ADD Coding Example
register, register	3	–	2	ADD CX, DX
register, memory	9 + EA	1	2-4	ADD DI, [BX],ALPHA
memory, register	16 + EA	2	2-4	ADD TEMP, CL
register, immediate	4	–	3-4	ADD CL, 2
memory, immediate	17 + EA	2	3-6	ADD ALPHA, 2
accumulator, immediate	4	–	2-3	ADD AX, 200

Conséquence : une addition se fait en plusieurs temps

- En langage évolué : addition de deux variables et résultat dans une troisième

  M3 = M1 + M2

- En langage d'assemblage :

  mov ax,[M1]

  add ax,[M2]

  mov [M3],ax

  ⇒ utilisation d'un registre interne pour le calcul appelé en général accumulateur, ici ax

Un contre-exemple

- L'instruction de transfert
  - mov [var_destination],[var_source] interdit

MOV	**MOV** destination, source Move			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
memory, accumulator	10	1	3	MOV ARRAY [SI], AL
accumulator, memory	10	1	3	MOV AZ, TEMP_RESULT
register, register	2	—	2	MOV AZ, CX
register, memory	8 + EA	1	2-4	MOV BP, STACK_TOP
memory, register	9 + EA	1	2-4	MOV COUNT [DI], CX
register, immediate	4	—	2-3	MOV CL, 2
memory, immediate	10 + EA	1	3-6	MOV MASK [BX] [SI], 2CH
seg-reg, reg 16	2	—	2	MOV ES, CX
seg-reg, mem 16	8 + EA	1	2-4	MOV DS, SEGMENT_BASE
reg 16, seg-reg	2	—	2	MOV BP, SS
memory, seg-reg	9 + EA	1	2-4	MOV [BX], SEG_SAVE, CS

Formats possibles d'une instruction

- Plusieurs champs dans l'instruction :
  - le champ instruction
  - éventuellement, un ou plusieurs champs opérande(s)

code op.				

cld

code op.	réf. opérande			

inc ax

code op.	réf. opérande	réf. opérande		

sub ax,bx

code op.	réf. opérande	réf. opérande	réf. opérande

Instruction

- Implications du format sur :
  - la taille et l'organisation mémoire
  - la structure du bus
- Implications du nombre sur :
  - ☺ une plus grande richesse du jeu d'instruction d'où l'appellation CISC (*Complex Instruction Set Computer*)
  - ☹ mais augmentation de la taille (*i.e.* surface) de la partie contrôle
- Des solutions :
  - le microprocesseur RISC (*Reduced-Instruction-Set Computer*)
  - le microprocesseur CRISC (*Complex-Reduced Instructions Set Computer*)

Autre classement (structure matérielle)

- CISC (*Complex Instruction Set Computer*)
- RISC (*Reduced-Instruction-Set Computer*)
- La synthèse : CRISC
  (*Complex-Reduced Instructions Set Computer*)
- VLIW (*Very Long Instruction Word*)
- Superscalaire
- Multi-cœurs
- Etc.

Conclusion

- Le processeur exécute sans fin son cycle d'exécution
- La mise à jour éventuelle des indicateurs binaires est faite à la fin de celui-ci