Architecture des ordinateurs

45 - Les débranchements (généralités)
Les sauts

Philippe Darche
IUT Paris Descartes

Le problème

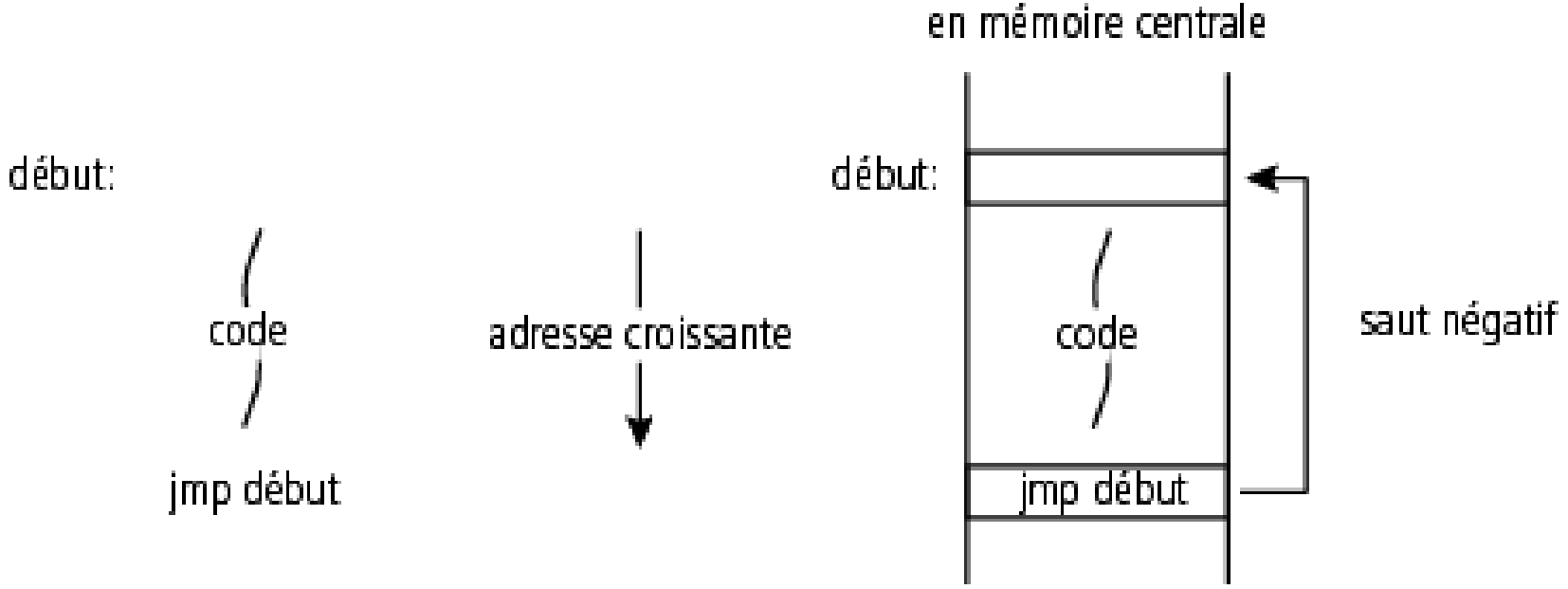
- Le modèle de machine de Von Neumann à la base :
  - exécution séquentielle des instructions
  - mémoire unifiée
    - codes instruction + données dans une même mémoire
- Comment faire pour exécuter une fonction ou une structure de contrôle ?
  - la réponse : il faut altérer le flot séquentiel d'exécution en utilisant, si possible, une condition

Les sous-questions

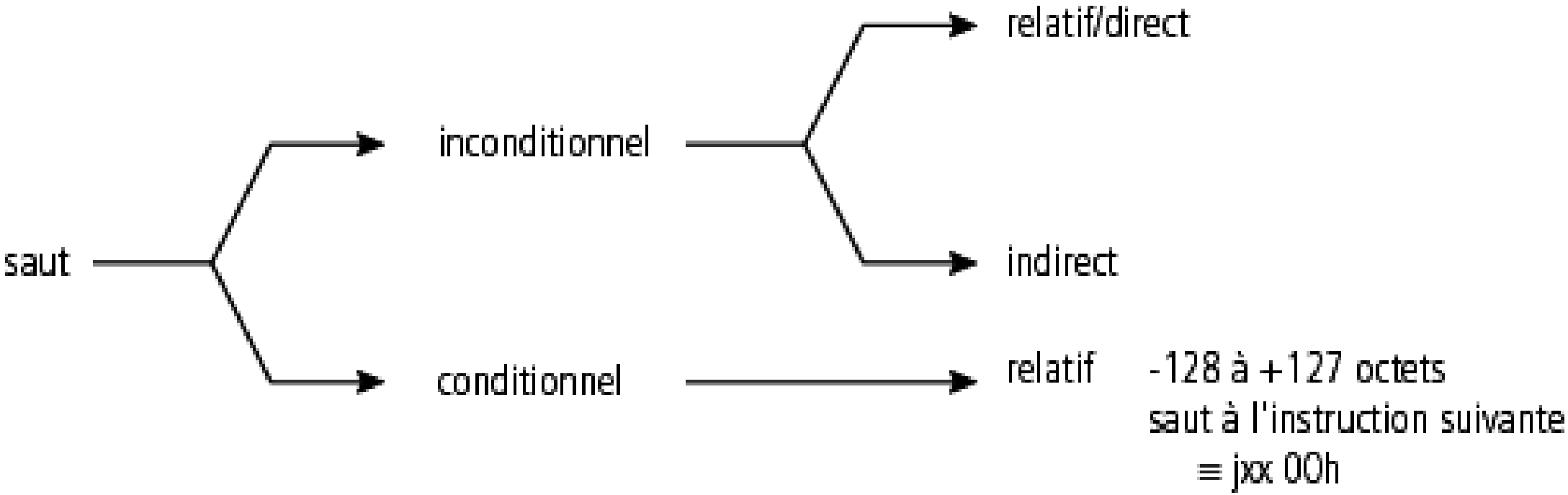
- Comment spécifier les adresses de saut et de retour (uniquement dans le cas de l'appel de sous-programme) en langage d'assemblage
- Quelle(s) condition(s) utilisée(s) ?

Les réponses pour les instructions de saut

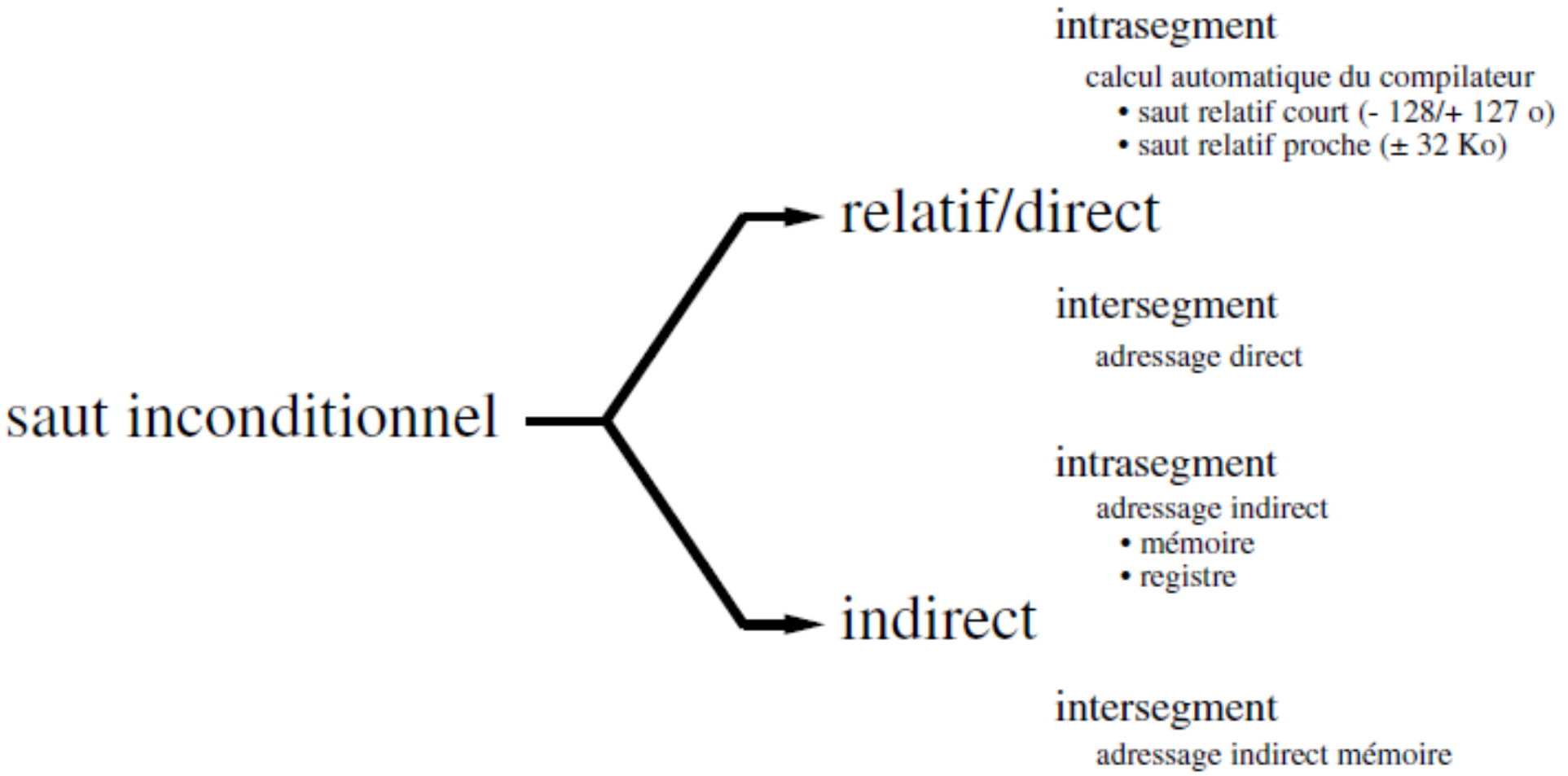
- L'adresse est spécifiée par un nom d'étiquette
  - exemple : une boucle infinie (un démon)
    - saut inconditionnel



Les type de saut



Les sauts inconditionnels



Saut inconditionnel direct

- Syntaxe :

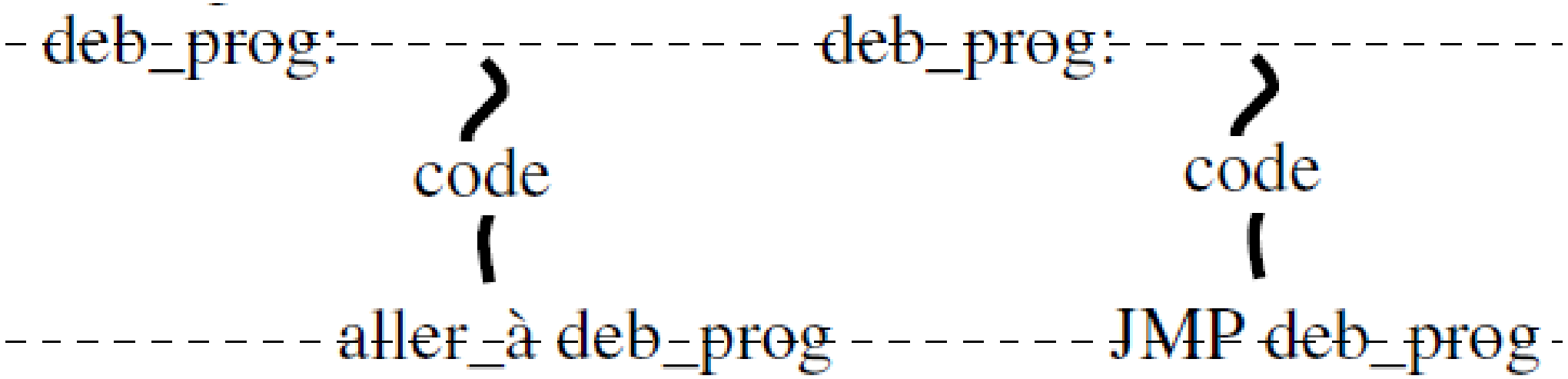
  jmp <nom_étiquette>

  ⇒ un opérande : étiquette
- Opération :
  - saut inconditionnel (= aller_à)
- Pseudo-code :

  si saut inter-segment (saut long)
  alors
  CS ← adresse-segment
  IP ← offset
  sinon
  IP ← (IP) + déplacement relatif
  fin si

Saut inconditionnel direct

- Exemple



Saut inconditionnel indirect

- Syntaxe :

  jmp <opérande>

  ⇒ un opérande : registre ou variable mémoire

- Opération :
  - saut inconditionnel (= aller_à)
- Pseudo-code :

  adressage registre

  (saut intra-segment uniquement)

  IP ← (registre)

Saut inconditionnel indirect (suite)

- Pseudo-code :

  adressage mémoire

  si saut inter-segment (saut long)

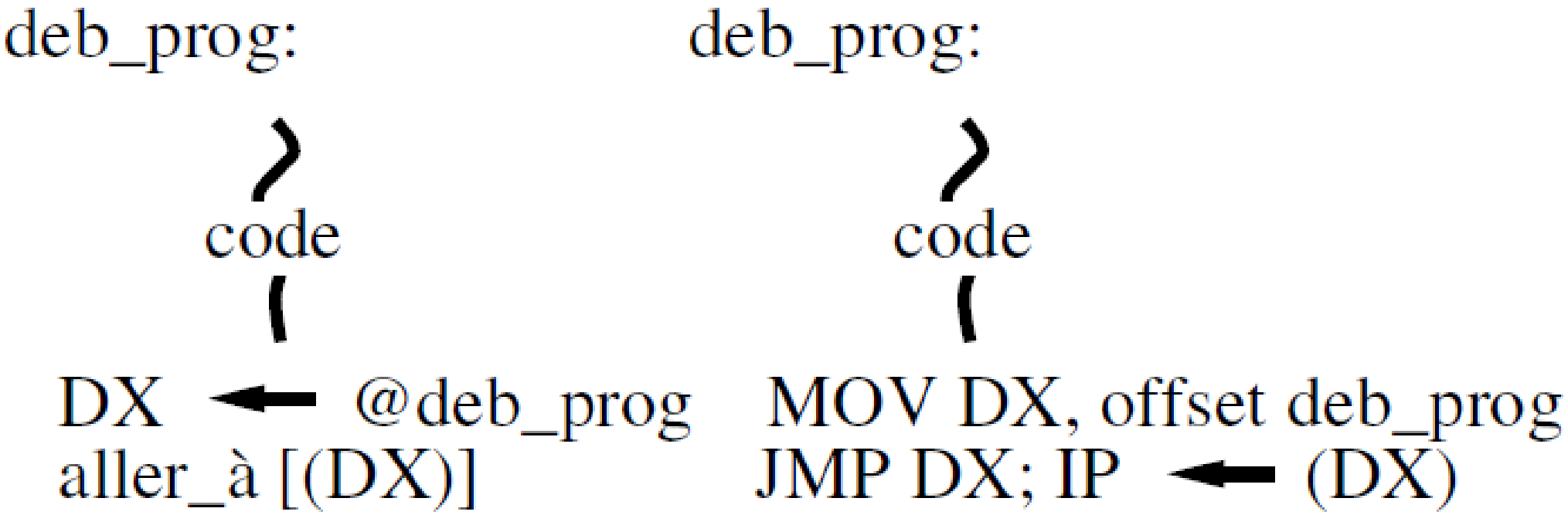
  alors CS ← adresse-segment

  fin si

  IP ← (opérande)

Saut inconditionnel indirect

- Exemple 1



Saut inconditionnel indirect

- Exemple 2

memo1 DW

deb_prog:

aller_à [(memo1)]

deb_prog:

MOV [memo1],offset deb_prog

JMP [memo1]; IP ← (memo1)

Saut conditionnel relatif

- Syntaxe :

  jxx <nom_étiquette>

  ⇒ un opérande, l'étiquette

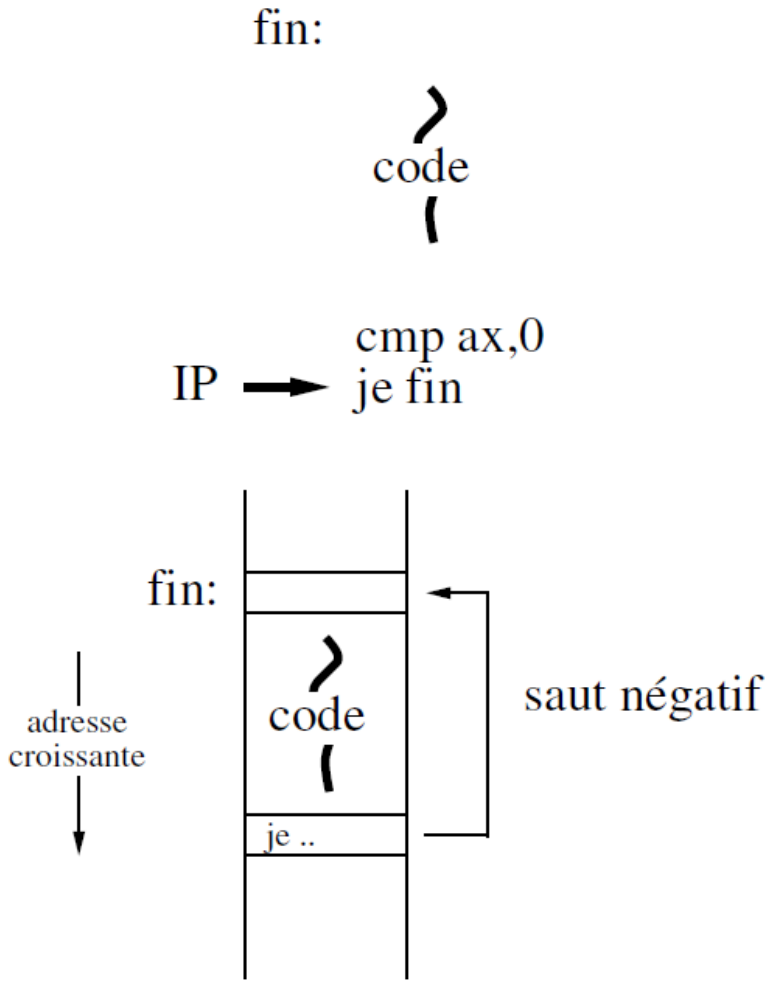
- Opération :
  - saut conditionnel
  - aller_à si <condition(s)> = vrai
- Pseudo-code :

  si conditions sur indicateur(s) binaire(s) = vrai

  alors IP ← (IP) + déplacement relatif

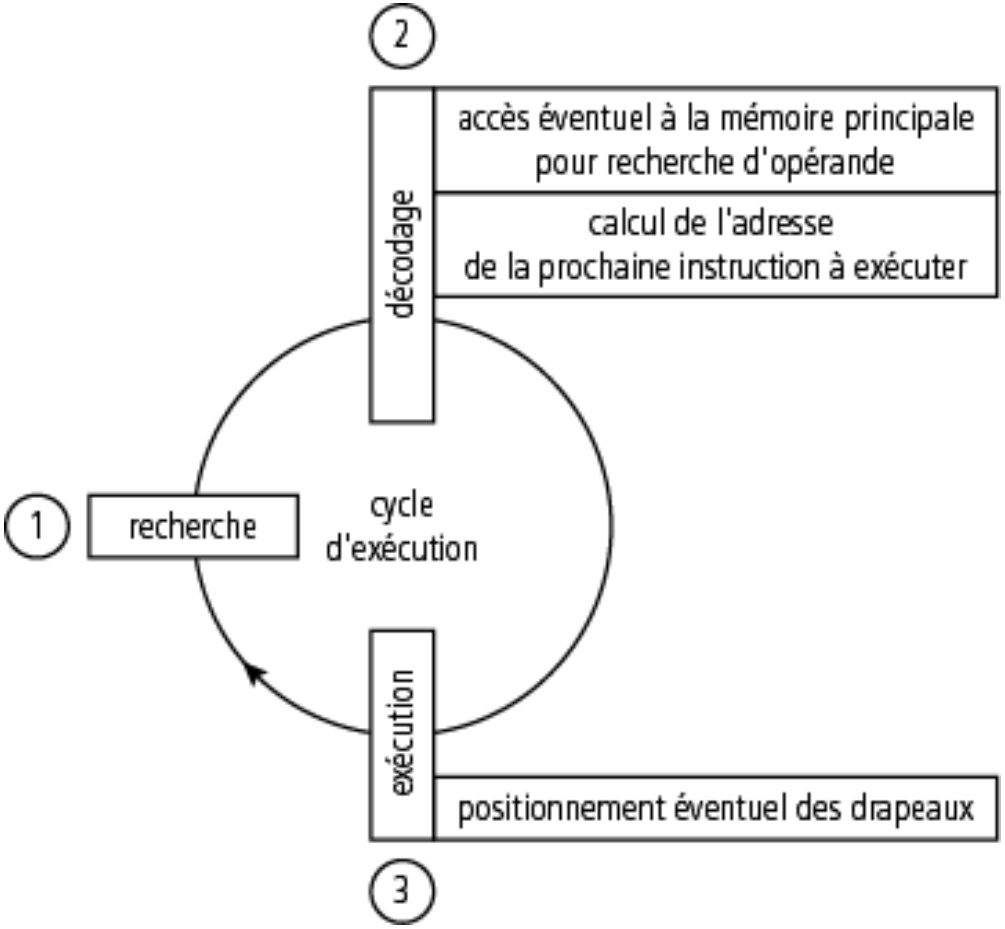
  fin si

Exemple de saut relatif



Cycle d'exécution de base (rappel)

- Un mouvement perpétuel ! décomposé en :
  - une étape de recherche
    - *fetch cycle*
  - une étape de décodage
    - *decode cycle*
  - une étape d'exécution
    - *execute cycle*



Le registre d'état du 8086 (rappel)

- OF (*Overflow Flag*) : indicateur de dépassement
- SF (*Sign Flag*) : indicateur de signe
- ZF (*Zero Flag*) : indicateur de zéro
- AF (*Half carry Flag*) : indicateur de demi-retenue
- PF (*Parity Flag*) : indicateur de parité
- CF (*Carry Flag*) : indicateur de retenue

Etats des indicateurs du registre d'état (1/2)

- Etat temporaire (*code condition*)
  - jusqu'à la prochaine instruction risquant de modifier l'indicateur binaire concerné
  - Renseignements sur l'état d'un résultat dont sa validité :
    - résultat nul, négatif, dépassement de format ou de capacité, etc.

Etats des indicateurs du registre d'état (2/2)

- Etat permanent
  - contrôle le fonctionnement du processeur
  - modification du comportement de certaines instructions
    - manipulation des caractères
  - modification du comportement du processeur
    - mode pas à pas
  - masquage des interruptions

⇒ Concerne l'UCo

Positionnement des indicateurs binaires

- Soit réalisé par la dernière instruction placée juste avant l'instruction de saut conditionnel
  - exemple : un `add` pour tester la retenue
- Soit réalisé par des instructions spécialisées dans la comparaison
  - instruction `cmp`
  - instruction `test`

La comparaison

- Syntaxe :

  cmp op_dest,op_source
  - ⇒ deux opérandes
- Opération :
  - soustraction du contenu de l'opérande destination avec le contenu de l'opérande source
  - pas de résultat, uniquement positionnement des indicateurs binaires
    - OF, SF, ZF, AF, PF et CF
- Pseudo-code :
  - (op_dest) - (op_source)
    avec positionnement des indicateurs binaires
- Utilité : comparaison

Le test

- Syntaxe :

  test op_dest,op_source

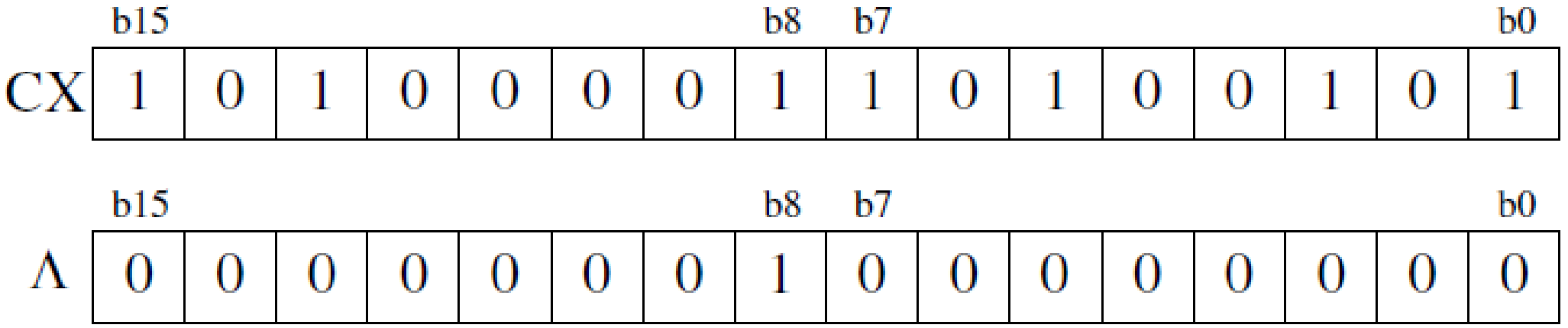
  ⇒ deux opérandes

- Opération :
  - ET logique du contenu de l'opérande destination avec le contenu de l'opérande source,
    pas de résultat mais uniquement positionnement des indicateurs binaires
- Pseudo-code :
  - (op_dest) • (op_source)
    avec positionnement des indicateurs binaires
- Utilité :
  - comparaison par rapport à zéro
  - test d'un bit particulier par masquage

Exemples

- test [memo],255

- mov cx,0A1A5h
- test cx,256; (cx) • 0100h (b8) = (b1)
- jz aucun_bit_même_position_à_1



Saut sur état d'indicateur

- Retenue (CF)
  - JNC (*Jump on Not Carry*)
  - JAE (*Jump on Above or Equal*)
  - JNB (*Jump on Not Below*)
    - CF = 0
  - JC (*Jump on Carry*)
  - JB (*Jump on Below*)
  - JNAE (*Jump on Not Above or Equal*)
    - CF = 1

Saut sur état d'indicateur

- Zéro (ZF)
  - JNE (*Jump on Not Equal*)
  - JNZ (*Jump on Not Zero*)
    - ZF= 0
  - JE (*Jump on Equal*)
  - JZ (*Jump on Zero*)
    - ZF= 1

Saut sur état d'indicateur

- Dépassement de capacité (OF)
  - JNO (*Jump on Not Overflow*)
    - OF = 0
  - JO (*Jump on Overflow*)
    - OF = 1
- Parité (PF)
  - JNP (*Jump on Not Parity*)
  - JPO (*Jump on Parity Odd*)
    - PF = 0
  - JP (*Jump on Parity*)
  - JPE (*Jump on Parity Even*)
    - PF = 1

Saut sur état d'indicateur

- Signe (SF)
  - JNS (*Jump on Not Sign*)
    - SF = 0
  - JS (*Jump on Sign*)
    - SF = 1

Saut sur état d'indicateur

- Représentation entière non signée ($\mathbb{N}$)
  - supérieur ($>$)
    - JA (*Jump on Above*)
    - JNBE (*Jump on Not Below or Equal*)
      - (CF and ZF) = 0
  - supérieur ou égal ($\geq$)
    - JAE (*Jump on Above or Equal*)
    - JNB (*Jump on Not Below*)
      - CF = 0
  - inférieur ($<$)
    - JB (*Jump on Below*)
    - JNAE (*Jump on Not Above or Equal*)
      - CF = 1

Saut sur état d'indicateur

- Représentation entière non signée (suite)
  - inférieur ou égal (≤)
    - JBE (*Jump on Below or Equal*)
    - JNA (*Jump on Not Above*)
      - (CF or ZF) = 1
  - non égal (≠)
    - JNE (*Jump on Not Equal*)
    - JNZ (*Jump on Not Zero*)
      - ZF = 0
  - égal (=)
    - JE (*Jump on Equal*)
    - JZ (*Jump on Zero*)
      - ZF = 1

Saut sur état d'indicateur

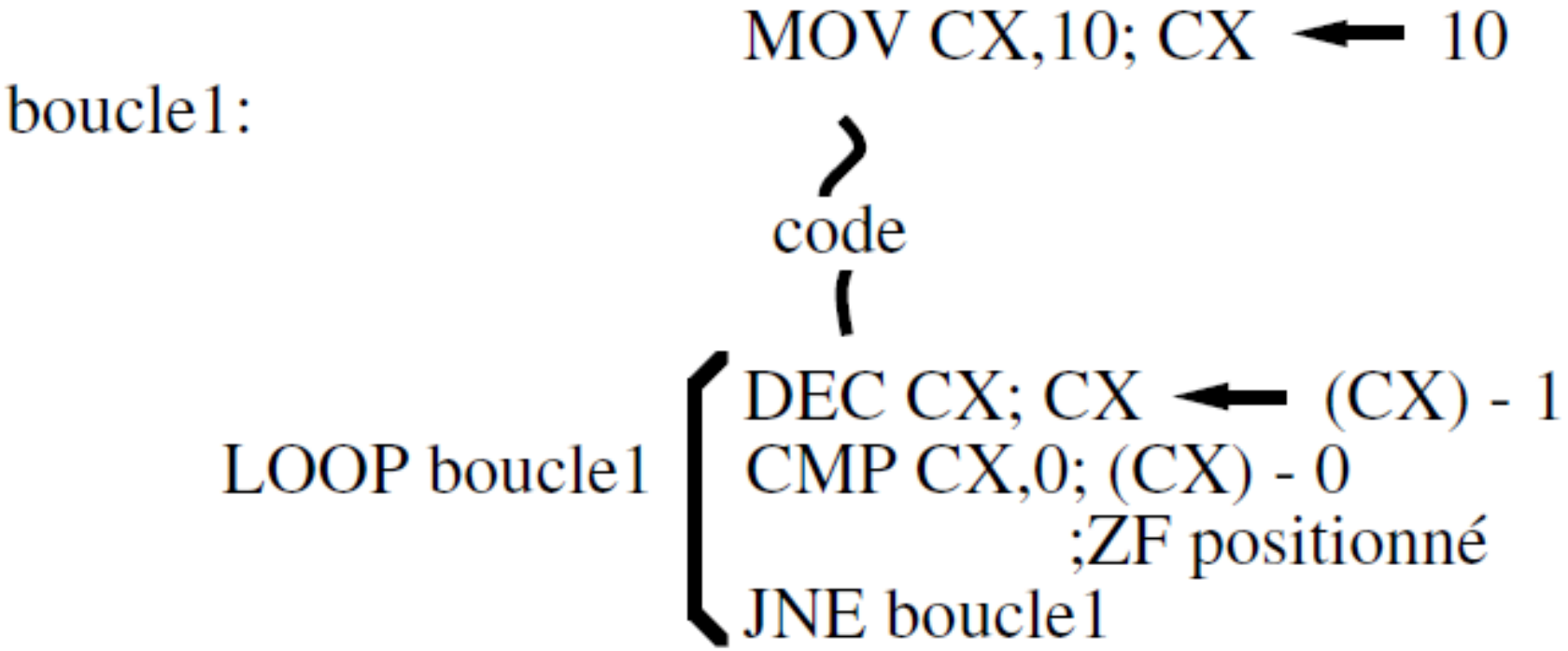
- Représentation entière signée ($\mathbb{Z}$)
  - supérieur ($>$)
    - JG (*Jump on Greater*)
    - JNLE (*Jump on Not Less or Equal*)
      - ((SF XOR OF) or ZF) = 0
        ou (SF = OF) or (ZF = 0)
  - supérieur ou égal ($\geq$)
    - JGE (*Jump on Greater or Equal*)
    - JNL (*Jump on Not Less*)
      - (SF XOR OF) = 0
        ou (SF = OF)

Saut sur état d'indicateur

- Représentation entière signée (suite)
  - inférieur (<)
    - JL (*Jump on Less*)
    - JNGE (*Jump on Not Greater or Equal*)
      - (SF XOR OF) = 1
        ou (SF ≠ OF)
  - inférieur ou égal (≤)
    - JLE (*Jump on Less or Equal*)
    - JNG (*Jump on Not Greater*)
      - ((SF XOR OF) or ZF) = 1
        ou (SF ≠ OF) or (ZF = 1)

Utilisation du saut conditionnel

- Implémentation des structures de contrôle
- Exemple : la boucle pour I de 1 à 10



Une instruction de haut niveau

- La boucle (instruction loop)
  - gestion automatique du compteur de boucle et du test
  - seule l'étiquette est spécifiée

  $\Rightarrow$ complique l'UCo

La boucle

- Syntaxe :

  loop <nom_étiquette>

  ⇒ un opérande

- Un registre implicite : CX
- Opération :
  - contrôle d'une structure "répéter jusqu'à" avec CX comme registre de comptage

La boucle (suite)

- Pseudo-code :

  CX ← (CX) - 1

  si (CX) ≠ 0

  alors IP ← (IP) + déplacement relatif (-128 à +127 octets)

  fin_si

- ou

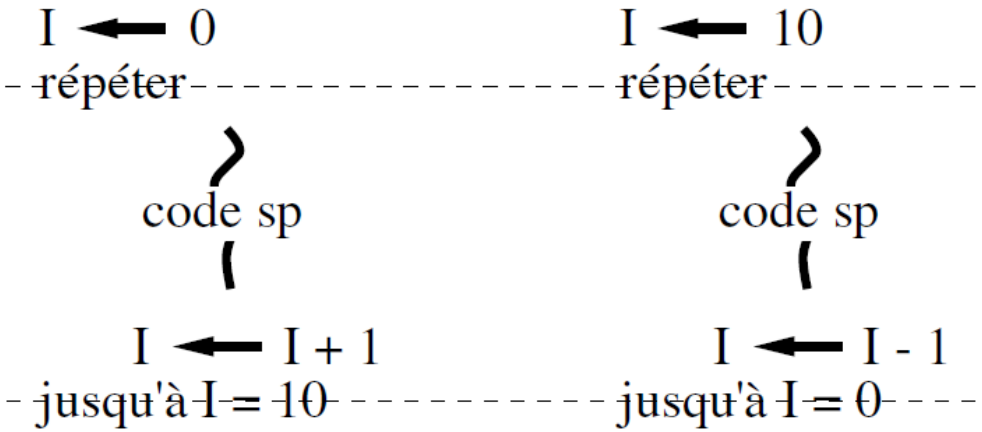
  CX ← (CX) - 1

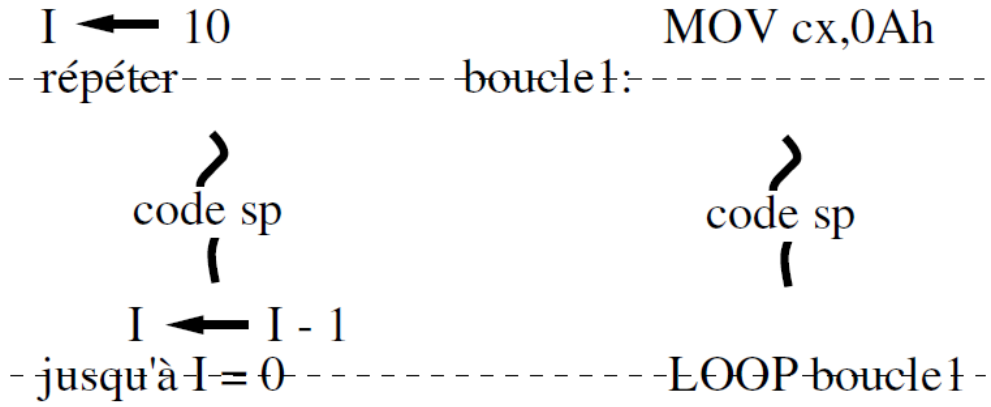
  si (CX) ≠ 0

  alors aller_à <nom_étiquette>

  fin_si

LOOP <nom_étiquette> - exemple





Conclusion

- Les sauts conditionnels et inconditionnels permettent de réaliser les structures de contrôle classiques des langages de haut niveau
  - si <condition> alors_sinon,
  - tant_que <condition> faire,
  - etc.
- ∃ une instruction de type « langage évolué » pour la boucle
- Voir TPs correspondants