Mise à jour: Février 2012

Architecture et Programmation

[Archi/Lycée]

Nicolas Bredèche

Maître de Conférences
Université Paris-Sud

bredeche@lri.fr

UNIVERSITÉ PARIS SUD

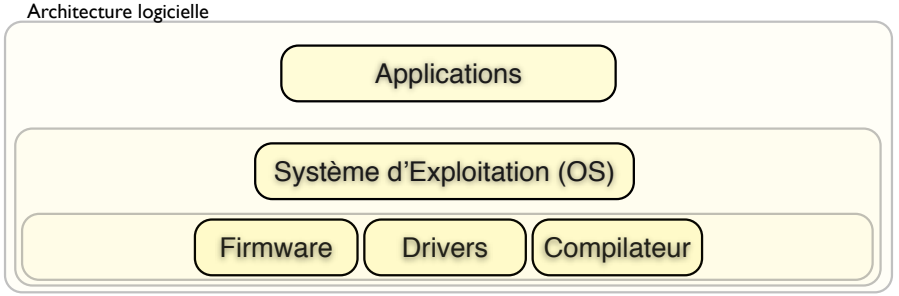
```
        .globl _start
BONJ:   .ascii  "Bonjour\n"
_start: mov     $4      , %eax
        mov     $1      , %ebx
        mov     $BONJ   , %ecx
        mov     $8      , %edx
        int     $0x80

        mov     $1      , %eax
        mov     $0      , %ebx
        int     $0x80
```

http://fr.wikipedia.org/wiki/Langage_assembleur

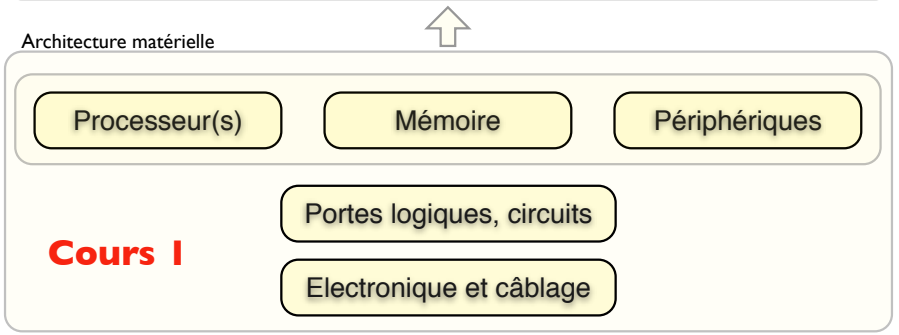
Ressources bibliographiques utilisées pour ce cours :

- Polycopié de cours de E.Viennet (Univ. Paris 13)
- http://wikipedia.com
- Patt, Patel. Introduction to Computing Systems: from bits and gates to C and beyond. McGraw Hill Int. Editions, 2001.

Architecture logicielle

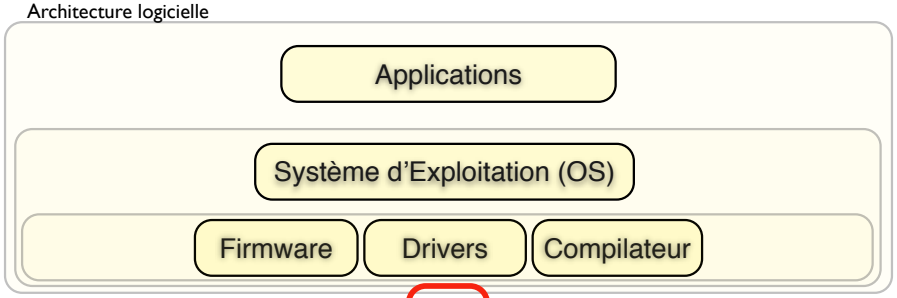


Architecture matérielle



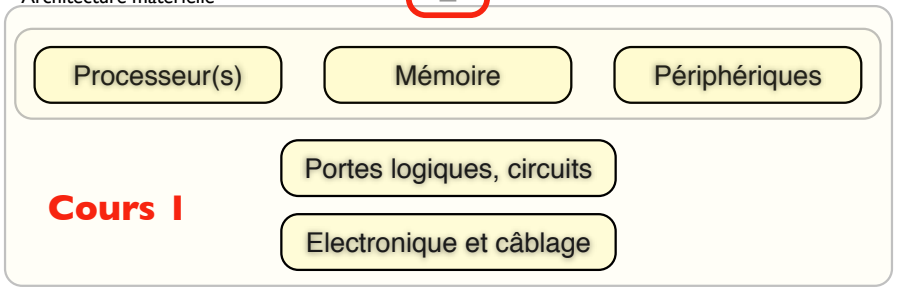
Cours 1

Architecture logicielle



Cours 2

Architecture matérielle

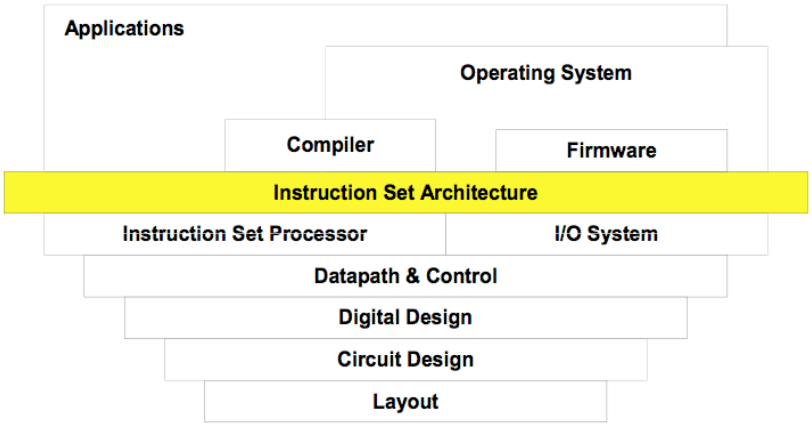


Cours 1

Architecture de Processeur

- Architecture de Processeur
- Micro-architecture
- Logique numérique

- Architecture externe de processeur «Instruction Set Architecture (ISA)»
  - Spécification fonctionnelle pour le programmeur
  - Eléments:
    - Jeu d'instructions
    - Registres visibles
    - Organisation de la mémoire
    - Organisation des E/S



source: David Paterons (depuis: M.Younis, CMCS 611)

Objectif du module

- Acquérir un...
  - Savoir
    - Architecture du point de vue de la programmation
    - Langage machine
    - Langage assembleur
  - Savoir faire
    - Ecrire un programme en assembleur

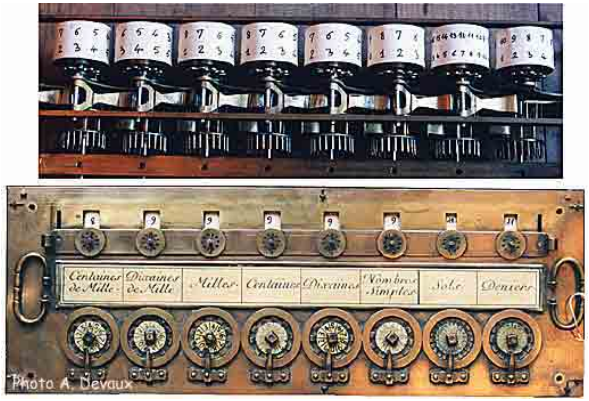
Plan du Cours #2

- Architecture, Langage Machine, Assembleur
  - Un peu d'histoire
  - Architecture de processeur
  - Langage Assembleur
  - Assembleur et Vie Artificielle

Brève histoire des ordinateurs et de leur programmation

La pascaline (1643)

addition, soustraction, conversion

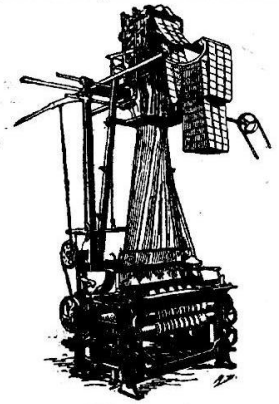


source de l'image: http://www.info.univ-angers.fr/~richer/

Métier à tisser Jacquard (1805)

utilisation de cartes perforées

JACQUARD (*ja-kar'*) n. Métier à tisser, inventé par Jacquard : *Un* JACQUARD. *Une* JACQUARD. ‖ On dit aussi MÉTIER À LA JACQUARD ou MÉTIER JACQUARD, et l'on écrit quelquefois *Jacquart* par un *t*. — ENCYCL. Les métiers *Jacquard* sont une des plus belles inventions de la mécanique industrielle. Ils ont considérablement étendu l'art du tissage, et ils ont permis d'exécuter avec facilité des articles qu'il aurait été impossible de fabriquer avec les anciens métiers. V. MÉTIER.

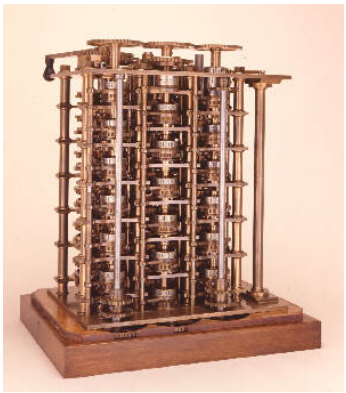


Métier Jacquard.

source: ?

Machine de Babbage (1833)

Ada Lovelace, première programmeuse du monde?



http://www.charlesbabbage.net/

Enigma (~1920)

Machine électro-mécanique pour le chiffrement

source: wikipedia

230 A. M. TURING [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

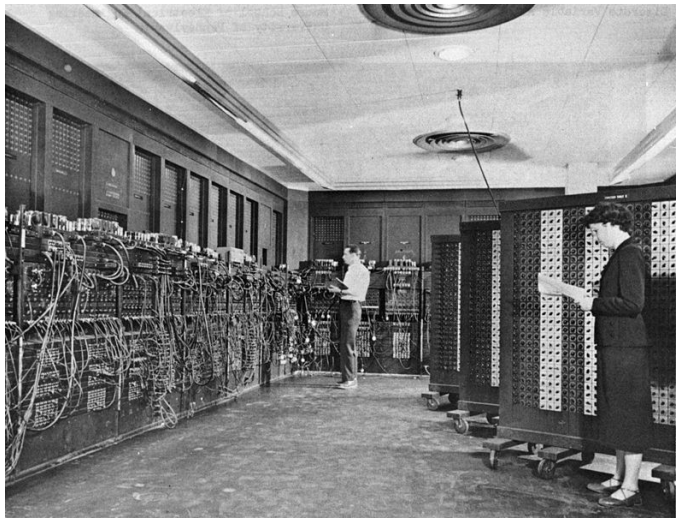
[Received 28 May, 1936.—Read 12 November, 1936.]

Description du premier calculateur universel programmable

Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

ENIAC (1946)

Les premiers ordinateur à tubes



source: wikipedia

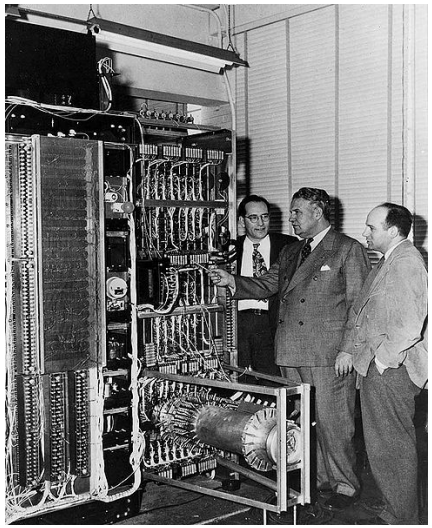
EDVAC (1948)

Adoption de l'architecture von Neumann

source: wikipedia

UNIVAC (1951)

abandon de la carte perforée pour la cassette



source: wikipedia

IBM 701 (1952)

calcul scientifique, pour l'armée

console opérateur

processeur

source: wikipedia

Historique 1/2

- 1946-1956 : Les ordinateurs à tubes
  - ENIAC, EDVAC, EDSAC, Mark 1, Univac, IBM 701
  - adoption de l'architecture von Neumann
- 1956-1963 : Les ordinateurs à transistors
  - le transistor est inventé en 1947
  - 1955: microprogrammation et jeu d'instructions
- 1963-1971 : Les circuits intégrés
  - circuit intégré, inventé en 1958
  - miniaturisation (permet la naissance du mini-ordinateur)
- 1971-...? : Le micro-processeur
  - on passe d'une histoire basée sur le matériel à une histoire basée sur le niveau des langages de programmation

Historique 2/2

- Perspective: langage de programmation
  - première génération : codage machine direct en binaire
  - deuxième génération : langage assembleur
  - troisième génération : langages évolués
    - Fortran, COBOL, Simula, APL, etc.
  - quatrième génération : langages évolués de 2ème génération
    - Pascal, C++, SQL, etc.
    - langages structurés, langages objets
  - *cinquième génération : langages et IA [échec pour l'instant]*
    - *lancé par le MITI au tout début des années 1980.*
    - *moteur d'inférence et Prolog*

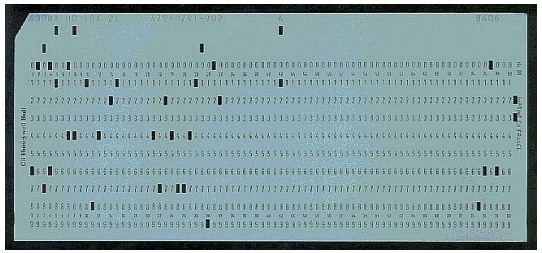
Historique 2/2

- Perspective: langage de programmation
  - première génération : codage machine direct en binaire
  - deuxième génération : langage assembleur

Chaque nouvelle génération de langage n'efface pas la précédente, mais s'empile sur les précédentes

  - quatrième génération : langages évolués de 2ème génération
    - Pascal, C++, SQL, etc.
    - langages structurés, langages objets
  - *cinquième génération : langages et IA [échec pour l'instant]*
    - *lancé par le MITI au tout début des années 1980.*
    - *moteur d'inférence et Prolog*

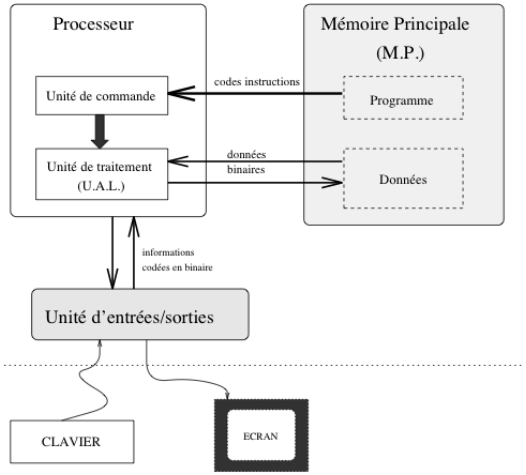
Synthèse



- Informatique = Information + Automatique
  - Ordinateur
    - Machine de traitement de l'information
    - Récupérer, stocker et traiter de l'information
      - texte, son, image, instructions
      - codage sous forme binaire

source: wikipedia

Architecture de Processeur

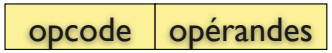


- Point de vue de la machine
  - Unité de commande
    - lecture en mémoire et décodage des instructions:
  - Unité de traitement (*ou* unité arithmétique et logique)
    - exécute les instructions qui manipulent des données
- Point de vue du programmeur
  - Un jeu d'instruction («*Instruction Set Architecture*» *(ISA)*)
  - +/- indépendant de l'implémentation
    - Exemple: le jeu d'instruction x86 et les processeurs Intel

Un jeu d'instructions

- Types d'instructions
  - calculs arithmétiques et logiques
  - écriture et lecture de données en mémoire
  - transfert de contrôle (tests et branchements)
- Types d'architecture
  - CISC, RISC
  - la nature des instructions varie

Une instruction



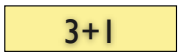
- Une instruction, c'est...
  - un «opcode» : son code opération
  - un ou des opérande(s) : ses paramètre(s)
- Une instruction à un coût
  - notion de cycles
    - lien avec la complexité de l'opération (ex.: division > addition)
    - lien avec le type d'objet manipulé (registre plus rapide que mémoire)
    - indépendant de la fréquence du processeur

Programme

- Un programme = des instructions élémentaires
  - actions très simples, exécutées par le processeur
    - opérations simples, écrire/lire la mémoire, sauts conditionnels
- Langage Machine (LM)
  - chaque instruction correspond à un code spécifique
  - directement exécutable par le processeur
- Assembleur (ASM)
  - un langage de bas niveau, mais lisible
  - instructions: symboles mnémotechniques
  - conversion directe en LM (réversible)

Illustration

langage de haut niveau



compréhensible pour le programmeur

assembleur

langage machine

compréhensible par la machine

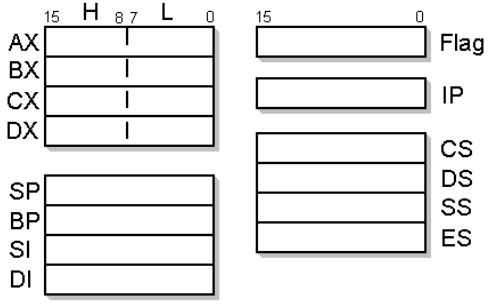
Mais ou met on le résultat?... il s'agit d'un exemple pour «jouer»

Que permet de voir l'architecture de processeur?

exemple avec le
jeu d'instructions x86

source: http://fr.wikipedia.org

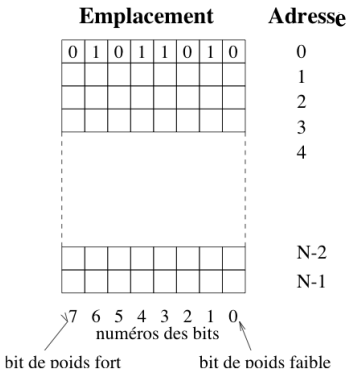
Registres



- Les registres
  - petites mémoires internes *très* rapides (notion de **mot**, dont la taille dépend du processeur)
  - le nombre de registres dépend de l'architecture
  - le registre «accumulateur» stocke le résultat des opérations *(ici: AX)*

http://fr.wikibooks.org/wiki/Programmation_Assembleur_x86/Registres

La mémoire



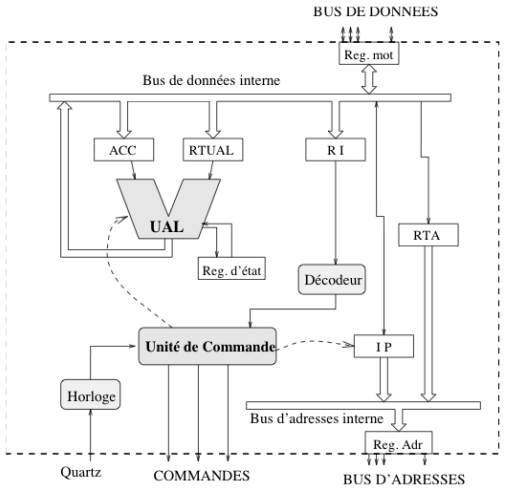
- On peut lire OU écrire sur la mémoire
  - pour lire, il faut une adresse (qui pointe sur un emplacement de **8 bits**)
  - pour écrire, il faut en plus une valeur (notion de **mot**, dont la taille dépend du processeur)

Polycopié d'Emmanuel Viennet (Univ. Paris 13)

La mémoire (cont.)

- Quelques remarques sur la mémoire:
  - 1 octet, c'est...
    - 8 bits
    - 1 byte (en anglais)
    - conversions:
      - 1 pétaoctet (Po) = 1 024 To = 2^50 octets
      - 1 téraoctet (To) = 1 024 Go = 2^40 octets
      - 1 gigaoctet (Go) = 1 024 Mo = 2^30 octets
      - 1 mégaoctet (Mo) = 1 024 ko = 2^20 octets
      - 1 kilooctet (ko) = 1 024 octets = 2^10 octets

Que fait l'unité de commande?



1 - lit l'instruction à exécuter
2 - effectue le traitement
3 - passe à l'instruction suivante

Assembleur

exemple avec le jeu d'instructions x86

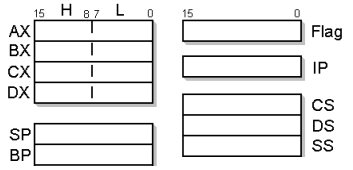
Langage Machine

```
A1 01 10 03 06 01 12 A3 01 14
```

Assembleur

```
Adresse  Contenu MP   Langage Symbolique   Explication en francais

0100     A1 01 10     MOV AX, [0110]       Charger AX avec le contenu
                                             de 0110.
0103     03 06 01 12  ADD AX, [0112]       Ajouter le contenu de 0112
                                             a AX (resultat dans AX).
0107     A3 01 14     MOV [0114], AX       Ranger AX en 0114.
```

Architecture x86 simplifiée



CPU 16 bits à accumulateur :
- bus de données 16 bits ;
- bus d'adresse 32 bits ;

Registres :
- accumulateur AX (16 bits) ;
- registres auxiliaires BX et CX (16 bits) ;

} *décomposable en deux sous-registres de 8 bits*

- pointeur d'instruction IP (16 bits) ;
- registres segments CS, DS, SS (16 bits) ;
- pointeur de pile SP (16 bits), et pointeur BP (16 bits).

Segment de code et de données

- Registre CS (code segment)
  - Le processeur lit le code à l'adresse CS:IP
  - Le processeur maintient à jour la valeur de IP
    - IP est initialisé par convention à l'adresse 100h
- Registre DS (Data Segment)
  - Ce registre pointe sur les données du programme
  - les instructions écrivent/lisent à l'adresse DS:<à définir>
    - exemple: MOV AX, [0110]
- Registre SS (Stack Segment)
  - cf. transparent sur la pile

Instructions

- écriture et lecture de données en mémoire
  - affectation mémoire vers registre (et l'inverse)
- calculs arithmétiques et logiques
  - logique: et, ou, ...
  - arithmétique simple: +,-,*,/
  - arithmétique complexe: sin, cos, tan
  - incrémentation, décrémentation, shift
- transfert de contrôle (tests et branchements)
  - comparaison
  - saut inconditionnel et sauts conditionnels
  - appel de fonction

Syntaxe d'affectation

```
MOV destination, source
```

MOV AX, *valeur*	AX ← *valeur*
MOV AX, [*adr*]	AX ← contenu de l'adresse *adr*.
MOV [*adr*], AX	range AX à l'adresse *adr*.

Adressage

- Adressage implicite

opcode	opérandes

  - uniquement un code opération
  - par exemple: «incrémenter AX»
- Adressage immédiat
  - code opération + opérande
  - par exemple: «ajouter <opérande> à AX»
- Adressage direct
  - l'opérande contient une adresse mémoire
  - par exemple: «écrire ds AX la valeur à l'adresse <opérande>»
- Adressage relatif
  - le code opération est une instruction de branchement
  - par exemple: «si AX = 0 alors IP = <opérande>»

Syntaxe d'adressage

Adressage immédiat

```
MOV AX, 12
```

Adressage direct

```
MOV AX, [A340]
```

Adressage relatif

```
JNE  0108
```


Calcul logique

OR ***destination, source*** *(OU)*

AND ***destination, source*** *(ET)*

XOR ***destination, source*** *(OU EXCLUSIF)*

```
OR   AX, FF00   ; AX <-  AX ou FF00
OR   AX, BX     ; AX <-  AX ou BX
OR   AX, [1492] ; AX <-  AX ou [1492]
```

exemple

Opérations arithmétiques

ADD AX, *valeur*	AX ← AX + *valeur*
ADD AX, [*adr*]	AX ← AX + contenu de *adr*.
SUB AX, *valeur*	AX ← AX - *valeur*
SUB AX, [*adr*]	AX ← AX - contenu de *adr*.
SHR AX, 1	décale AX à droite.
SHL AX, 1	décale AX à gauche.
INC AX	AX ← AX + 1
DEC AX	AX ← AX - 1

et aussi ROL, ROR, MUL, DIV...

Branchement inconditionnel

JMP <adresse>

```
Adresse   Contenu MP   Langage Symbolique    Explication en francais

0100      B8 00 00     MOV AX, 0             met AX a zero
0103      A3 01 40     MOV [140], AX         ecrit a l'adresse 140
0106      EB FC        JMP 0103              branche en 103
0107                   xxx -> instruction jamais executee
```


Branchements conditionnels

- Registre d'état
  - Il est mis à jour pour certaines opérations
    - opérations arithmétiques
    - instruction CMP (CoMPare)
    - instructions STC (SeT Carry) et CLC (CLear Carry)
  - Il contient les «indicateurs»
    - Zero Flag (ZF)
    - Carry Flag (CF)
    - Sign Flag (SF)
    - Overflow Flag (OF)
  - Il n'est pas accessible directement
  - Il est utilisé par les instructions de branchement

Branchements conditionnels (cont.)

JE *Jump if Equal*
saut si ZF = 1 ;

JNE *Jump if Not Equal*
saut si ZF = 0 ;

JG *Jump if Greater*
saut si ZF = 0 et SF = OF ;

JLE *Jump if Lower or Equal*
saut si ZF=1 ou SF≠OF ;

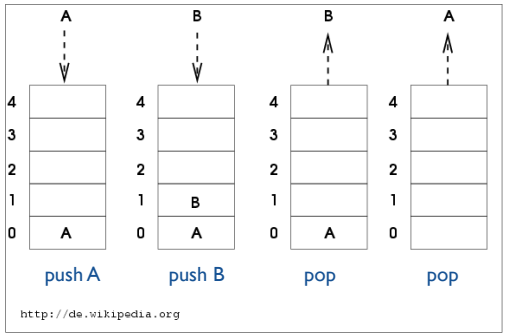
JA *Jump if Above*
saut si CF=0 et ZF=0 ;

JBE *Jump if Below or Equal*
saut si CF=1 ou ZF=1.

JB *Jump if Below*
saut si CF=1.

La pile

mov AX, 0xA
push AX
mov AX, 0xB
push AX
pop
mov BX,AX
pop



Il s'agit d'une pile «LIFO»
cf. registre SS (Stack Segment) et SP (Stack Pointer)

Les interruptions

```
MOV  AH, 4C
INT 21
```

Retour au DOS

Deux exemples


```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```



```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```

```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```



```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```



```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```



```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```

```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```



```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```



```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```



```
; Programme calculant la somme de deux entiers de 16 bits

data        SEGMENT
A           DW  10      ; A = 10
B           DW  1789    ; B = 1789
Result      DW  ?       ; resultat
data        ENDS

code        SEGMENT

            ASSUME DS:data, CS:code

debut:      MOV  AX, data ; etiquette car 1ere instruction
            MOV  DS, AX   ; initialise DS

            ; Le programme:
            MOV  AX, A
            ADD  AX, B
            MOV  result, AX ; range resultat

            ; Retour au DOS:
            MOV  AH, 4CH
            INT  21H
code        ENDS

            END debut ; etiquette de la 1ere inst.
```

Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```


Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```


Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```


Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```

Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```


Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```


Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```


Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```

Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```


Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```


Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```

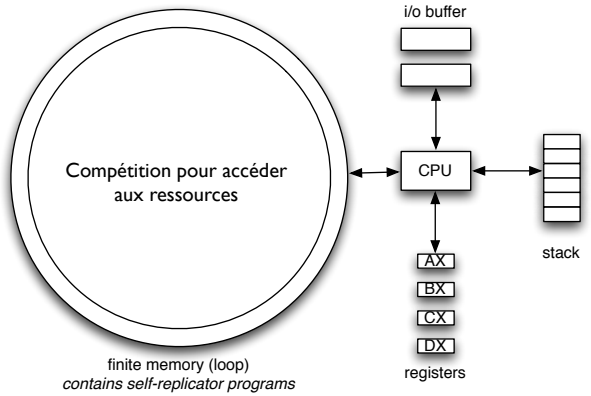

Programme convertissant les majuscules en minuscules

```
data        SEGMENT
tab         DB 'Un boeuf Bourguignon', '$'
data        ENDS

code        SEGMENT
            ASSUME DS:data, CS:code

debut:      MOV  AX, data
            MOV  DS, AX

            MOV  BX, offset tab ; adresse debut tableau

repet:      MOV  AL, [BX]       ; lis 1 caractere
            AND  AL, 11011111b  ; force bit 5 a zero
            MOV  [BX], AL       ; range le caractere
            INC  BX             ; passe au suivant
            CMP  AL, '$'        ; arrive au $ final ?
            JNE  repet          ; sinon recommencer

            MOV  AH, 4CH
            INT  21H            ; Retour au DOS
code        ENDS
            END debut
```

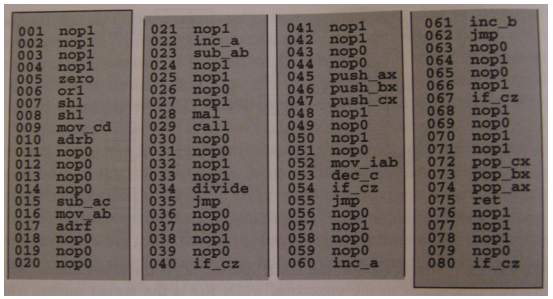
Synthèse

- Assembleur
  - Assembleur et Langage Machine (bijectif)
  - Un point de passage obligé
    - compilation d'un langage de haut niveau au langage machine
- Et aussi...
  - Optimisation...
    - des opérations: réduire le nombre cycles
    - des compilateurs: mieux traduire
  - Machine virtuelle (ex.: Java)
    - bytecode, portabilité, compilation à la volée
  - Code embarqué, firmware
    - mémoire, vitesse... mais moins facile à programmer

Vie Artificielle et Assembleur



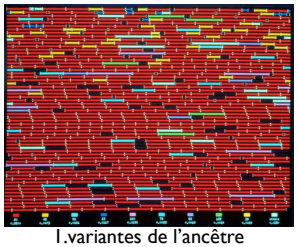
TIERRA
Thomas Ray, 1988



Le programme ancêtre
(extrait de Artificial Life, 1994, C. Adami)

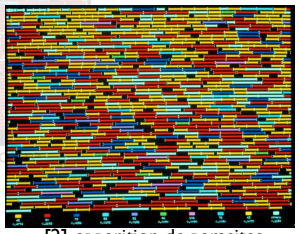
1. le programme "ancètre"
   début - reproduction - procédure de copie - fin
   un pointeur par programme
2. mutation possible lors de la duplication
3. The Reaper: supprime le plus vieux (ie. lent) si besoin de place (ressources CPU et mémoire limitée)

source: adami98



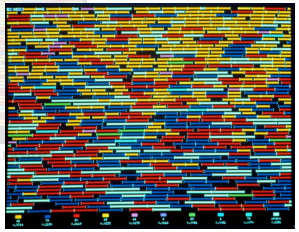
1.variantes de l'ancêtre
(en rouge)

cf. http://life.ou.edu/ (images: Marc Cygnus)



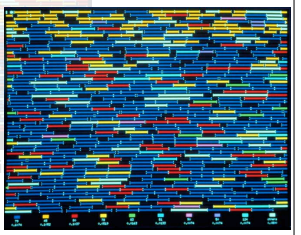
[2]. apparition de parasites
(en jaune)

cf. http://life.ou.edu/ (images: Marc Cygnus)



[3]. apparition d'hôtes immunisés
(en bleu)

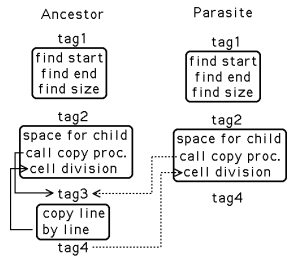
cf. http://life.ou.edu/ (images: Marc Cygnus)



[4]. les hôtes immunisés se multiplient.

cf. http://life.ou.edu/ (images: Marc Cygnus)

Propriétés émergentes



TIERRA : les premiers parasites

- Dynamiques de l'évolution
  - Course aux armements
  - Equilibres ponctuées
- Phénomènes observés
  - Compression, hôtes-parasites, symbioses

source: adami98

Fin du cours #2