Architecture des ordinateurs

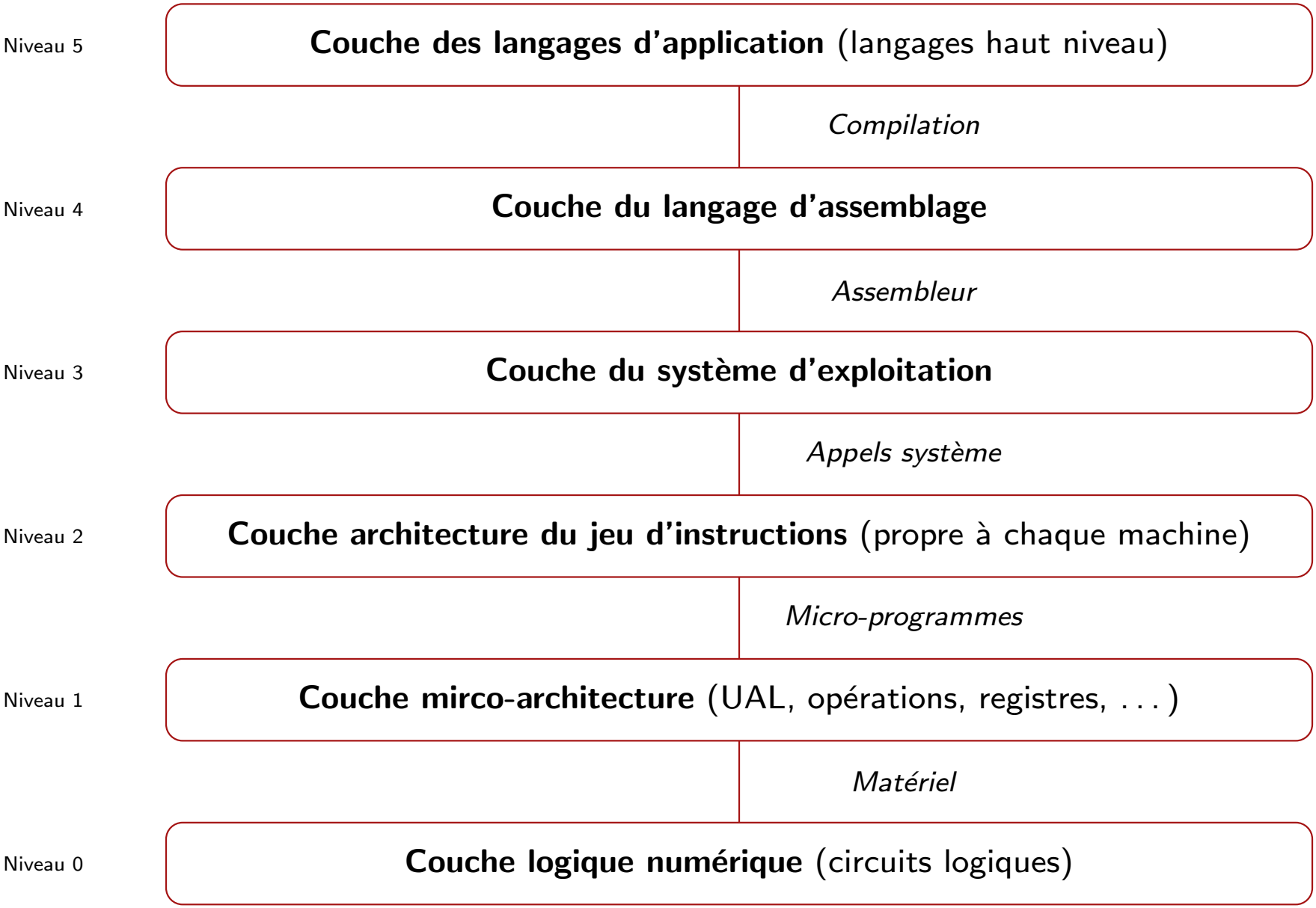
Micro-architecture

L1 MIASHS

UFR Mathématiques et Informatique

(2014 - 2015)

Rappel : architecture en couches



Motivations

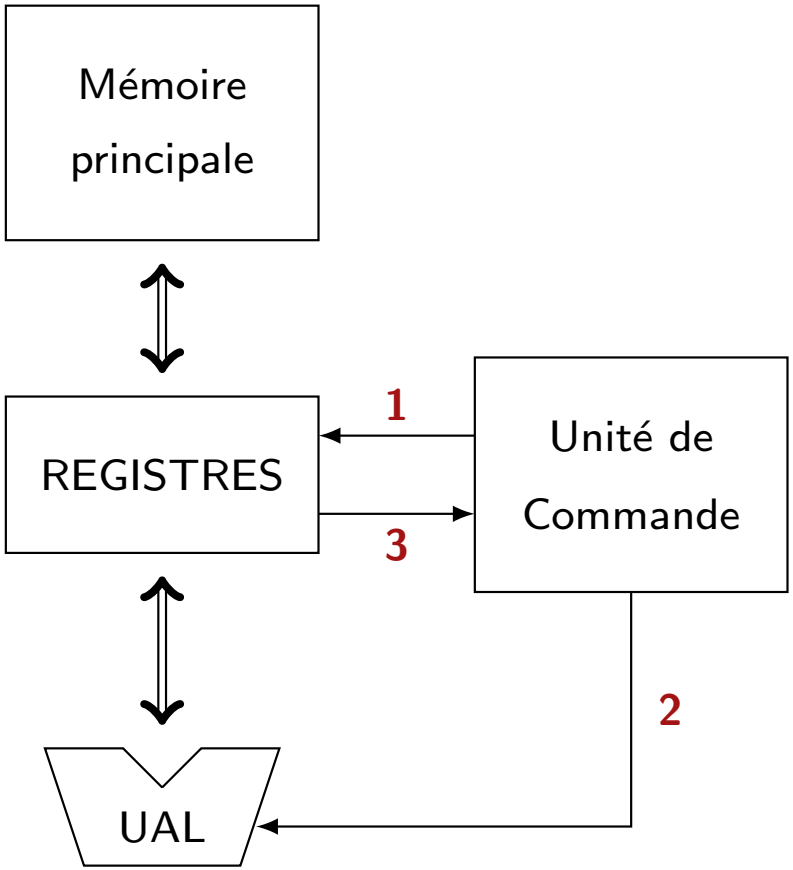
- Comment assembler les différents circuits vus dans les cours précédents pour fabriquer un processeur ?
- Comment interagir avec la mémoire ?
- L'UAL est le "cerveau" de la machine.
  - Comment peut-on la commander ?
  - Quel langage utilise-t-on pour communiquer avec l'UAL ?
- Comment de simples opérations réalisables par l'UAL peuvent-elles aboutir à un programme ?

$\Rightarrow$ Il n'y a pas de réponse universelle, pas de modèle d'architecture.

$\Rightarrow$ Compromis entre coût et performances.

Chemin des données

Comment les différents éléments présents dans le processeur interagissent-ils ?



1. L'UC active certains registres pour :
   - lire en mémoire
   - écrire en mémoire
   - transférer des données vers l'UAL
   - transférer des données depuis l'UAL
2. L'UC commande l'action de l'UAL
3. L'état des registres permet de choisir la prochaine commande

Micro-architecture (1/3)

- Implémente le jeu d'instructions spécifié par la Couche ISA supérieure
- S'appuie sur la Couche Logique inférieure
- Dépend :
  - du jeu d'instructions à implémenter :
    - CISC (*Complex Instruction Set Computer*)
    - RISC (*Reduced Instruction Set Computer*)
  - du coût et des performances

Micro-architecture (2/3)

L'exécution d'une instruction peut se décomposer en plusieurs sous-étapes :

1. Recherche (*Fetch*)

   → Récupération de la prochaine instruction à exécuter

2. Décodage (*Decode*)

   → Détermination du type et de la nature des opérandes

3. Exécution (*Execute*)

   → Mise en œuvre des unités fonctionnelles

4. Terminaison (*Complete*)

   → Modification en retour des registres ou de la mémoire

Micro-architecture (3/3)

Micro-architecture = **problème de programmation**

- Chaque instruction du niveau ISA est une fonction
- Le programme maître (micro-programme) :
  - Boucle infinie
  - Détermine à chaque tour la bonne fonction à appeler et l'exécute
  - Dispose de variables d'état : compteur ordinal, registres généraux, ...
    - ⋆ Accessibles par chacune des fonctions
    - ⋆ Modifiées spécifiquement selon la nature de la fonction

Micro-programme

Micro-instruction : mot binaire codant des signaux de commande

Micro-code : ensemble de micro-instructions stockées en mémoire ROM

Instruction : bloc de micro-instructions réalisant "une opération simple"

- Instructions de déplacement : `MOV`, `XCHG`, `...`
- Manipulation de la pile : `PUSH` et `POP`
- Instructions arithmétiques et logiques : `ADD`, `SUB`, `AND`, `OR` et `NOT`
- Sauts et boucles : `JUMP`, `Jxx`, `LOOPxx`, `...`
- Appels de fonctions : `CALL` et `RET`

Gestion de la mémoire (1/2)

Problème n°1 : Calculer $((1+2)\times(3+4))+((5+6)\times(7+8))$
en mémorisant les calculs intermédiaires

- Combien faut-il de registres dans ce cas là ?
- Quel est le nombre maximal de registres utilisés dans un calcul ?

Problème n°2 : Stocker les variables locales des fonctions

- Où sont conservées les variables locales des fonctions ?
- Solution simple : attribuer à toutes les variables des adresses fixes différentes
- Problème : si une fonction en appelle une autre ou s'appelle elle-même ?

Gestion de la mémoire (2/2)

Procédures avec **paramètres d'appel** et **variables locales**

- Variables **accessibles uniquement pendant l'exécution** de la procédure
- Ne peuvent pas résider à une adresse absolue en mémoire

$\Rightarrow$ Nécessité de **créer dynamiquement** des instances de ces variables

$\Rightarrow$ **Suppression** des variables **à la fin de l'exécution** de la procédure

La Pile (1/3)

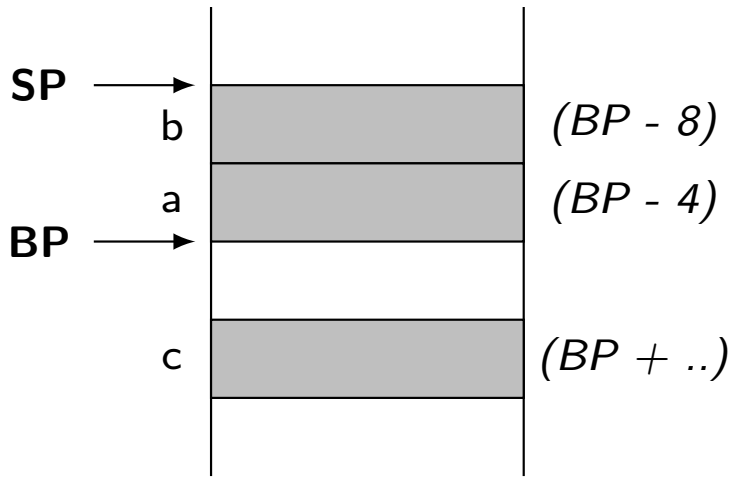
- Zone de la mémoire **accessible uniquement relativement à des registres**
- Gérée au moyen de deux registres :
  - Un **registre de base** (*Base Pointer* (BP))
    $\rightarrow$ Pointe sur le début de la zone mémoire allouée pour les variables locales de la procédure courante
  - Un **registre de sommet de pile** (*Stack Pointer* (SP))
    $\rightarrow$ Pointe sur le dernier mot mémoire alloué
- **LIFO** (*Last In First Out*)
- Opérations :
  - `PUSH` (ajout)
  - `POP` (retrait)

La Pile (2/3)

- Les paramètres et les variables locales à la procédure courante sont **référencées par rapport à la valeur courante de BP**

- La zone de données **référencée par BP et limitée par SP** est appelée **“contexte courant”**

```
ALGO monAlgo
ENTREES
  c : NUMERIQUE
VARIABLES
  a : NUMERIQUE
  b : NUMERIQUE
DEBUT
  ...
  monAlgo(a+1)
  ...
FIN
```

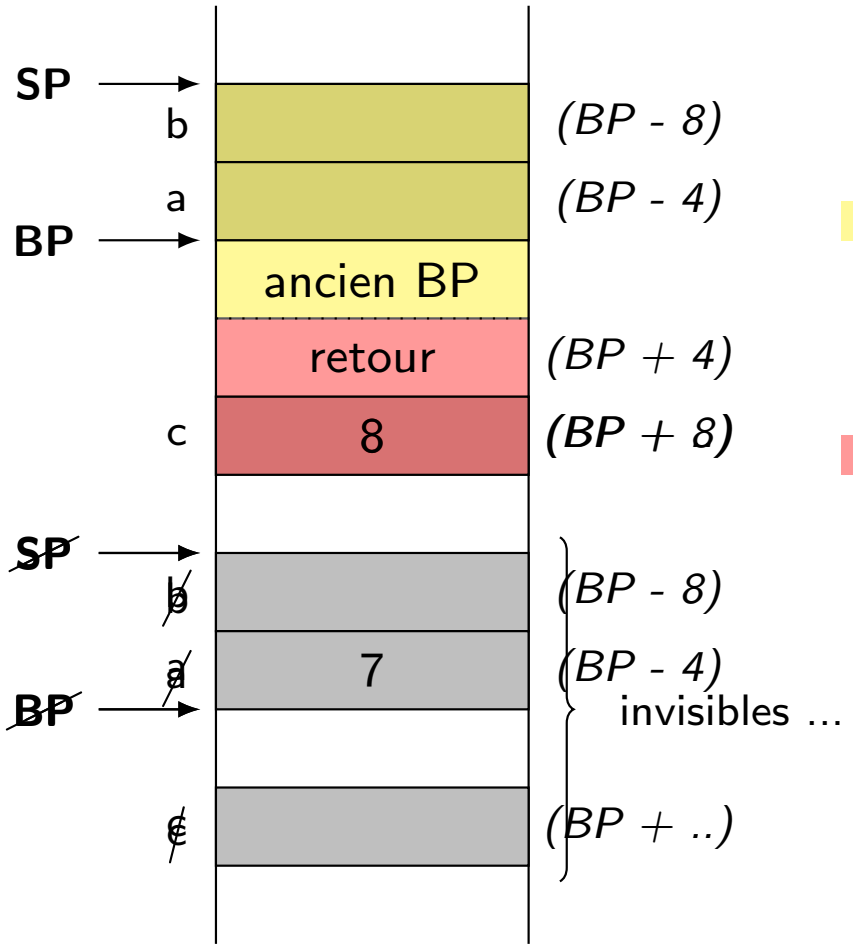


La Pile (3/3)

Lors d'un appel de procédure :

- un nouveau contexte courant se crée au sommet de la pile
- on sauve l'ancien BP dans la pile

```
ALGO monAlgo
ENTREES
  c : NUMERIQUE
VARIABLES
  a : NUMERIQUE
  b : NUMERIQUE
DEBUT
  ...
  monAlgo(a+1)
  ...
FIN
```



Gérée par la procédure appelée

Gérée par la procédure appelante

Séquence d'appel d'une procédure (1/2)

- Partie gérée par la procédure appelante :
  - Empilage des paramètres listés dans la procédure
  - Appel de la procédure (**sauvegarde automatiquement l'adresse de retour dans la pile**)
- Partie gérée par la procédure appelée (début de procédure) :
  - Empilage de l'**ancien BP dans la pile**
  - Copie de **la valeur de SP dans celle de BP**
    - ⋆ Le nouveau contexte est basé à la position courante de SP
    - ⋆ Le premier paramètre est accessible à l'adresse de BP plus la taille de deux adresses entières (l'ancien BP et l'adresse de retour), donc (BP+8)
  - Soustraction à SP de la taille des variables locales

Séquence d'appel d'une procédure (2/2)

- Partie gérée par la procédure (fin de procédure) :
  - Remise dans SP de la valeur de BP

    $\rightarrow$ Libère la zone des variables locales à la procédure
  - Dépilement de BP

    $\rightarrow$ BP pointe de nouveau sur le contexte appelant
  - Appel de l'instruction de retour

    $\rightarrow$ Dépile la valeur de retour située dans la pile

Séquence de retour d'une procédure

- Gérée par la procédure appelante
- Incrémentation de SP de la taille de tous les paramètres empilés avant l'appel de procédure
- Retour complet à l'état antérieur

Exemple d'instruction arithmétique : ADD (1/2)

Rôle :

- "retirer" les deux éléments au sommet de la pile,
- les additionner,
- et placer le résultat au sommet de la pile

1ère étape : récupérer les mots au sommet de la pile

- Placer SP dans le Registre d'Adresse mémoire (RA)
- Placer dans le Registre de Données mémoire (RD) le mot pointé par RA
- Recopier RD dans le Registre Tampon (RT)
- Calculer l'adresse du mot juste "en dessous" du sommet de la pile et la placer dans RA, ainsi que dans le regsitre SP (supprime le premier mot à additionner)

Exemple d'instruction arithmétique : ADD (2/2)

2ème étape : calculer la somme et la placer au sommet de la pile

- Placer dans RD le mot pointé par RA
- Additionner RD et RT
- Mettre le résultat dans RD
- Écrire en mémoire le contenu de RD à l'adresse contenue dans RA (à la place du second mot à additionner)

Exemple d'instruction arithmétique : `PUSH var`

Rôle : mettre la variable locale `var` au sommet de la pile

1ère étape : récupérer la variable locale désignée par `var`

- Placer BP dans le RT
- Calculer l'adresse de la variable locale
  et mettre le résultat dans RA
- Placer dans RD le mot pointée par RA

2ème étape : placer la variable locale au sommet de la pile

- Calculer l'adresse du nouveau sommet de pile et la placer dans RA
  ainsi que dans le registre SP
- Écrire le contenu de RD à l'adresse mémoire pointée par RA

Performances

Lors de la conception d'une micro-architecture, plusieurs paramètres entrent en considération, notamment :

- la rapidité
- le coût

$\Rightarrow$ Trouver des compromis : par exemple, rajouter des registres pour accéder rapidement à plus de données, mais pas trop car les registres coûtent très cher.