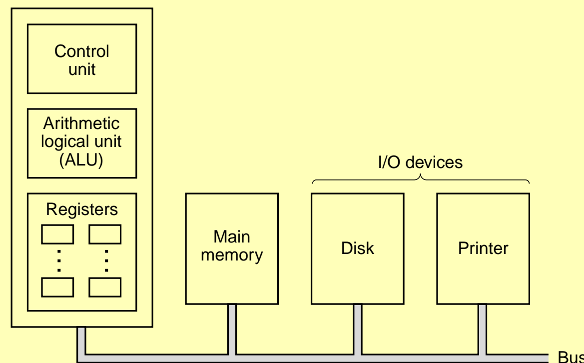
Livre de référence : Architecture de l'ordinateur, Andrew Tanenbaum, 4ème édition

Plan

- Vue globale
- Le processeur
- La mémoire
  - Principale
  - Cache
  - Secondaire
- Les entrées / sorties
- En quelques mots

Abordé ultérieurement

Vue globale de l'ordinateur



Le processeur

- CPU (Central Processing Unit)
  - Ou UC (Unité Centrale)
- Exécute les programmes stockés en mémoire principale (Fetch – Decode – Execute)
- Constitué de
  - Unité de commande (contrôle)
  - Unité arithmétique et logique (UAL – ALU)
  - Une « mémoire locale » : les registres

Les registres du processeur (1/2)

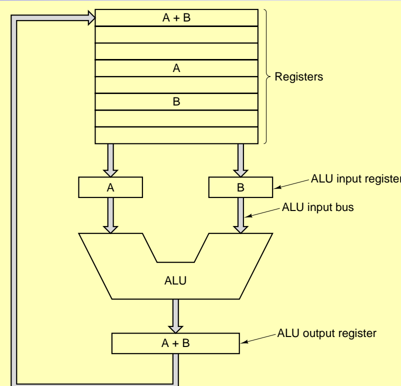
- Registres dédiés :
  - PC (Program Counter) ou CO (Compteur Ordinal)
    - ✓Contient l'adresse de la prochaine instruction à exécuter.
  - RI (Instruction Register) ou RI (Registre d'instruction)
    - ✓Contient le code de l'instruction que le processeur est en train d'exécuter.
  - Flags ou Drapeaux
    - ✓Diverses informations sur l'état du processeur.

Les registres du processeur (2/2)

- Registres utilisateur
  - L'accumulateur
    - ✓ADD R7 (ACC:=ACC+R7)
  - Registres généraux
    - ✓LD R3,100 (R3:=MEM[PC+100])
  - Registres de manipulation d'adresse
    - ✓ST R7,10(R3) MEM[R3+10]=R7
  - Registres pour le calcul en virgule flottante

L'UAL et le chemin de données

- Exemple
  - 3 registres :
    - ✓A, B et RESULT
- Le type d'opération et les registres impliqués sont définis par l'unité de commande.



Deux types d'instruction

- Registre – Registre
  - L'instruction porte sur des données qui sont déjà situées dans des registres et le résultat doit également être écrit dans un registre.
- Registre – Mémoire (ou E/S)
  - L'instruction a besoin de lire (les opérandes) ou écrire (les résultats) en mémoire ou sur les E/S.

L'exécution d'une instruction

- Charger la prochaine instruction depuis la mémoire principale (à l'adresse PC) vers le RI.
- Modifier le compteur ordinal pour qu'il pointe sur l'instruction suivante à exécuter (sauf branchement)
- Décoder l'instruction
- Localiser en mémoire les données nécessaires à l'instruction
- Charger ces données dans les registres
- Exécuter l'instruction
- Recommencer le cycle

L'interpréteur (1/2)

```
public class Interp {
    static int PC;
    static int AC;
    static int instr;
    static int instr_type;
    static int data_loc;
    static int data;
    static boolean run_bit = true;
```


L'interpréteur (2/2)

```
public static void interpret(    int memory[],
                                 int starting_address) {
PC = starting_address;
while (run_bit) {
  instr = memory[PC];
  PC=PC+1;
  instr_type=get_instr_type(instr);
  data_loc = find_data(instr,instr_type);
  if (data_loc >= 0) data = memory[data_loc];
  execute(instr_type,data);
}
```


Un mini-ordinateur dans le processeur

- Le processeur est en fait un mini-ordinateur qui exécute un micro-programme qui ne fait intervenir que les registres et l'UAL.
- Ce micro-programme est utilisé pour interpréter les programmes situés à l'extérieur du processeur (dans la mémoire principale).
- En changeant le micro-programme, on change le jeu d'instruction du processeur.

L'interprétation / l'exécution

- L'interprétation offre :
  - UAL très simple (bon marché)
  - ISA configurable
  - Beaucoup de micro-opérations sur des registres donc le temps d'accès est rapide
- L'exécution offre
  - La rapidité due à une UAL capable de réaliser des opérations complexes en moins de cycles
- La lenteur des mémoires externes a joué en faveur des systèmes interprétés

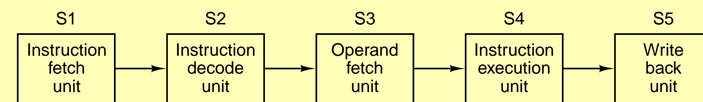
Conception des ordinateurs modernes

- Toute instruction est traitée directement par le matériel (UAL), comme dans le RISC
  - Certains processeurs disposent néanmoins des deux technologies combinées (RISC / CISC)
- Maximiser la vitesse d'exécution (MIPS)
  - Augmenter le parallélisme
  - Utiliser des instructions simples à décoder
- Minimiser les accès mémoire
  - Seulement deux instructions (LOAD / STORE)
  - Utiliser beaucoup de registres

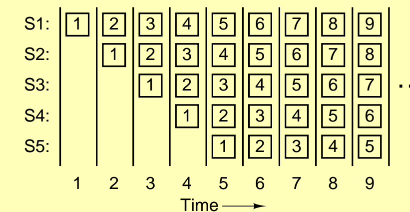
Le parallélisme

- Au niveau des instructions
  - L'idée consiste à exécuter plusieurs instructions d'un même programme en même temps. Cela implique une analyse (statique ou dynamique) des dépendances entre les instructions
- Au niveau du processeur
  - On utilise plusieurs processeurs pour exécuter un même programme mais chaque processeur travaille sur des données différentes.

Le pipeline



(a)

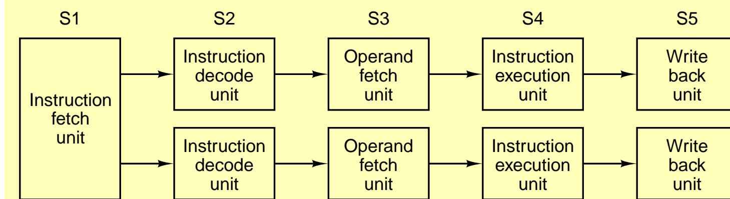


(b)

Performances du Pipeline

- Si un étage dure 2 ns, il faut 10 ns pour qu'une instruction passe dans tous les étages.
  - En CISC, cela fait 100 MIPS
  - En RISC (avec le pipeline), on obtient un résultat toutes les 2ns, donc la performance est de 500 MIPS
- Ces performances chutent avec les aléas qui obligent de « vider » le pipeline
  - Branchement
  - Séquence d'instructions dépendantes
  - Interruptions …

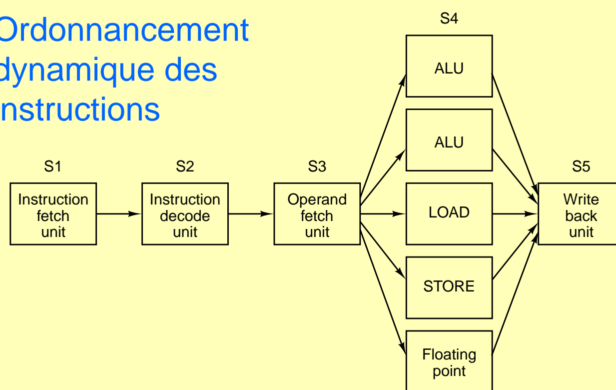
L'architecture superscalaire (1/2)



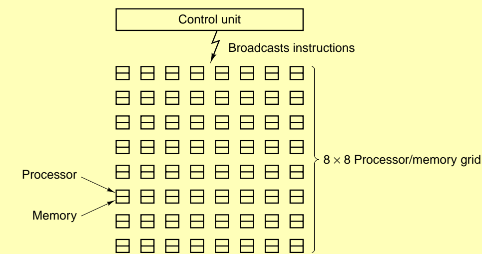
- Deux (ou plus …) pipelines exécutent en parallèle les instructions compatibles.
- La gestion de l'accès aux registres devient plus complexe

L'architecture superscalaire (2/2)

- Ordonnancement dynamique des instructions



Ce processeur matriciel x64 les performances



- Un seul programme est exécuté par les 64 processeurs sur des données différentes

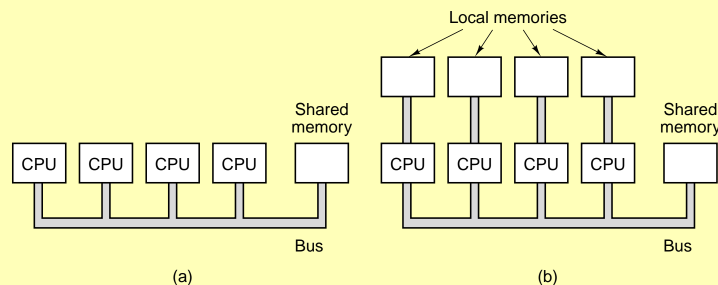
Le processeur vectoriel

- Vu par l'utilisateur, il a les mêmes propriétés que le processeur matriciel.
- Son architecture est radicalement différente :
  - Un seul additionneur pipeline est utilisé pour réaliser les opérations sur les vecteurs
  - C'est son jeu d'instructions (CISC) et son unité de calcul spécialisée qui lui permet de manipuler directement des vecteurs
  - Il est moins performant que le processeur matriciel mais il est aussi beaucoup moins cher.

Le multiprocesseur (1/2)

- Plusieurs processeurs exécutent leur propre programme.
- Les données (en mémoire principale) sont communes ou partiellement communes
- Gestion de l'accès au bus (matériel)
- Gestion de l'accès aux données (logiciel)

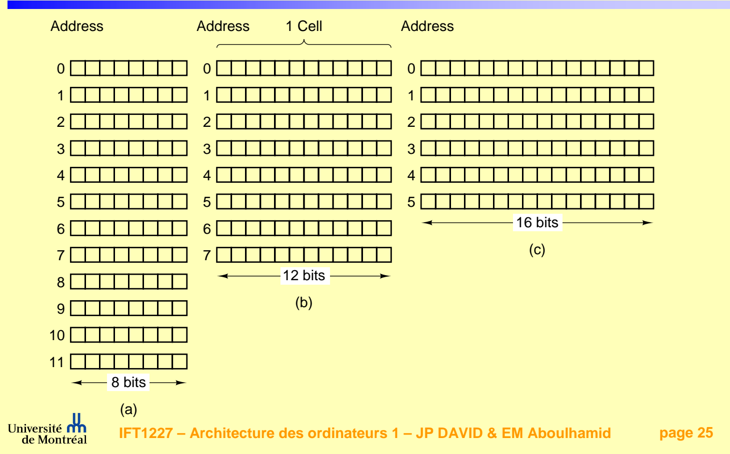
Le multiprocesseur (2/2)



Organisation des bits dans la mémoire

- Soit une mémoire de 96 bits. Cela signifie qu'elle contient en tout 96 cases réparties dans des tiroirs :
  - 12 tiroirs de 8 cases
  - 8 tiroirs de 12 cases
  - 6 tiroirs de 16 cases
  - 16 tiroirs de 6 cases
  - 3 tiroirs de 32 cases
  - 32 tiroirs de 3 cases …et les adresses ?

Exemple de 3 organisations différentes

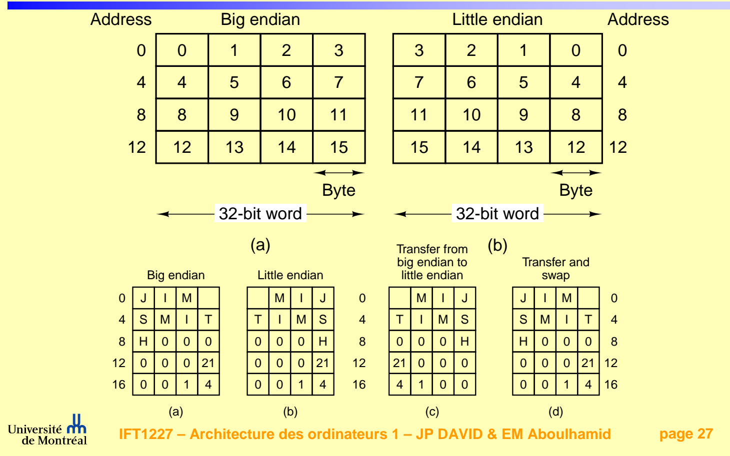


Organisation des octets

- Dans le mode « gros-boutiste » (big-endian), le poids fort du mot de 32 bits est mémorisé à l'adresse la plus faible.
- Dans le mode « petit-boutiste » (little-endian), le poids faible du mot de 32 bits est mémorisé à l'adresse la plus faible.

C'est une simple convention mais cela pose des soucis de compatibilité de données entre deux systèmes qui ont des modes différents

Le mode gros-boutiste/petit-boutiste



La détection d'erreurs

- Distance de Hamming :
  - Nombre d'inversion de bits entre 2 mots :
    - ✓ Exemple : A = 11100000 et B = 10100011
    - ✓ A XOR B = 01000011, soit une distance de 3
- Le principe de la détection d'erreur consiste à augmenter artificiellement la distance entre 2 quelconques mots de données en ajoutant des bits supplémentaires
  - Pour des mots de 8 bits, on peut ajouter un bit de parité :
    - ✓ Soit 256 combinaisons licites/512 et la distance de Hamming minimale entre deux mots vaut 2

Les codes correcteurs d'erreurs

- Pour détecter des changements sur n bits, il faut une distance de Hamming de n+1 entre tous les mots licites.
- Pour détecter et corriger des changements sur n bits, il faut une distance de Hamming de 2n+1 entre tous les mots licites.
  - Ex, n=5, D=11
  - Si un mot subit 5 inversions de bit, il sera à une distance 5 du mot d'origine et minimum 6 de tous les autres. On peut donc le retrouver en cherchant le seul et unique mot donc la distance est inférieure ou égale à 5.

Le code de Hamming

- Soit un mot de N bits numérotés $b_0..b_{N-1}$
- On crée un nouveau mot de N+P bits numérotés $H_0..H_{P-1}$
- H_i est un bit de parité pour les ^2 (1,2,4 …)
- H_i est un bit de donnée b_j sinon.
- Soit X, l'expression binaire de i
  - H_i participe au codage de parité de H_j si et seulement si X_j a un '1' commun avec H_i

Illustration du code de Hamming

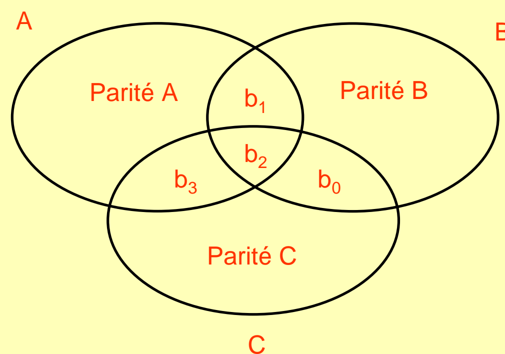
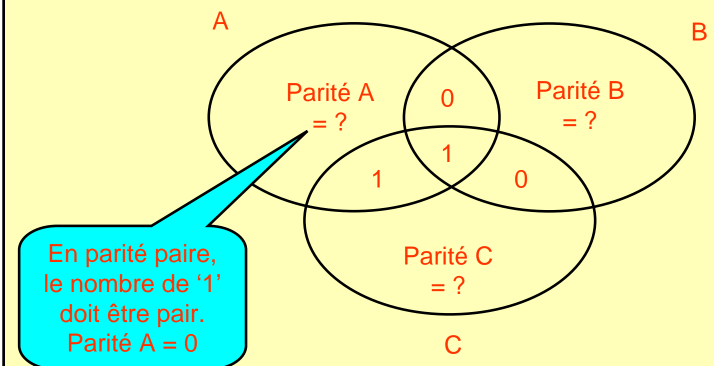
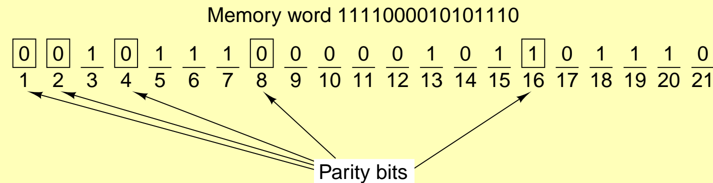


Illustration du code de Hamming



Exemple du code de Hamming

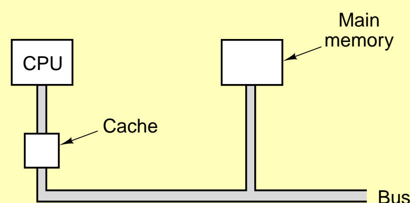


- Le bit 4 se code en binaire : 0100, il sert donc à calculer la parité de :00101 (5), 00110 (6), 00111 (7),01100 (12), 01101 (13), 01110 (14), 01111 (15),10100 (20), 10101 (21)

Principe de la mémoire cache

- Plus les données sont éloignées des ressources de calcul, plus il faut du temps pour aller les chercher
  - Registre : très court (1 cycle)
  - Mémoire principale : (trop) long (ex : 10 cycles)
- Principe de localité : la plupart des données utilisées au temps t+1 sont peu éloignées de celles utilisées au temps t (exemple : les instructions d'un programme se suivent (sauf exception), une multiplication matricielle utilises deux tableaux de données localisés …)
- La mémoire cache permet de garder tout près de l'unité de calcul les données qui ont été utilisées récemment pour y accéder plus rapidement

Architecture à mémoire cache



c : temps d'accès à la mémoire cache

m : temps d'accès à la mémoire principale

h : le taux de succès (n^bre de succès/n^bre d'accès)

$T_{acc} = c + (1-h)m$