Chap. 2 - Structure d'un ordinateur

2.1 Processeur ou unité centrale (CPU)

- 2.1.1 Organisation du CPU
- 2.1.2 Exécution d'une instruction
- 2.1.3 RISC versus CISC
- 2.1.4 Principes de conception des ordinateurs
- 2.1.5 Parallélisme des instructions
- 2.1.6 Parallélisme du processeur

2.2 Mémoire principale

Bits, Adresses mémoire, Organisation des octets, Codes correcteurs d'erreurs, Mémoires cache, Conditionnement physique

2.3 Mémoires secondaires

Hiérarchie de mémoire, Disques durs, Disques souples, Disques IDE, SCSI, RAID, CD-x, DVD-x

2.4 Entrée / Sortie (E/S)

Bus, Terminaux, Souris, Imprimantes, Modems, Codification des caractères

2.1 Processeur ou unité centrale (CPU)

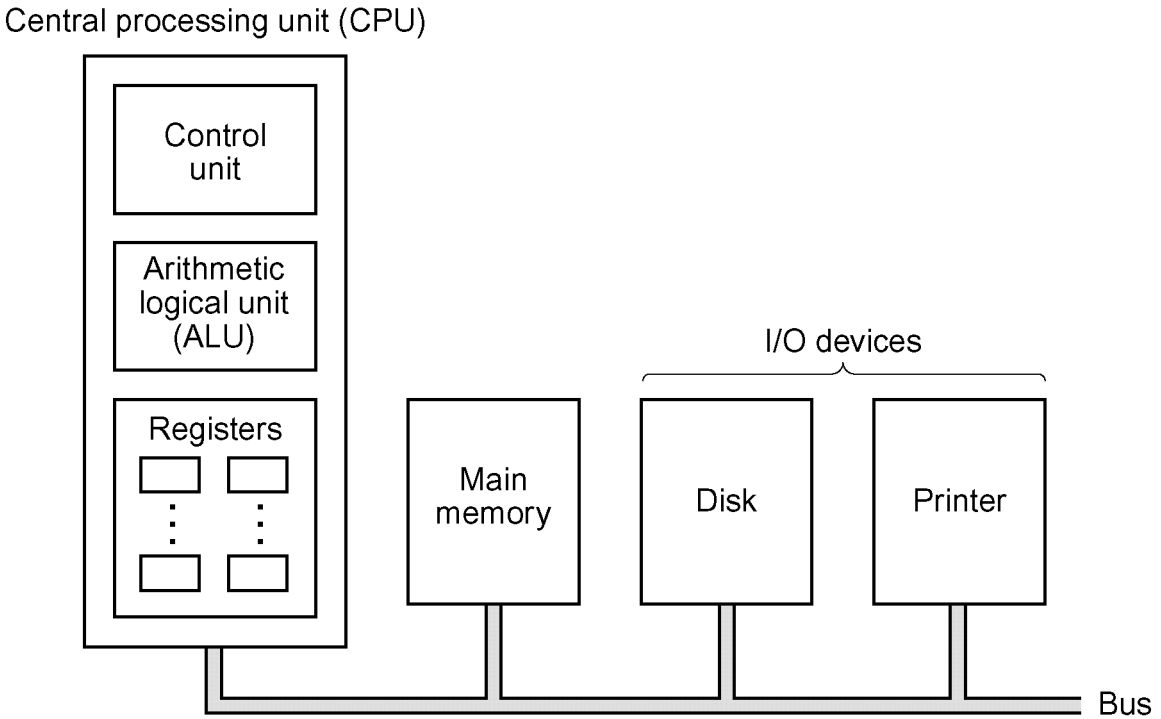

Figure 2-1. The organization of a simple computer with one CPU and two I/O devices.

2.1 Processeur ou unité centrale (CPU)

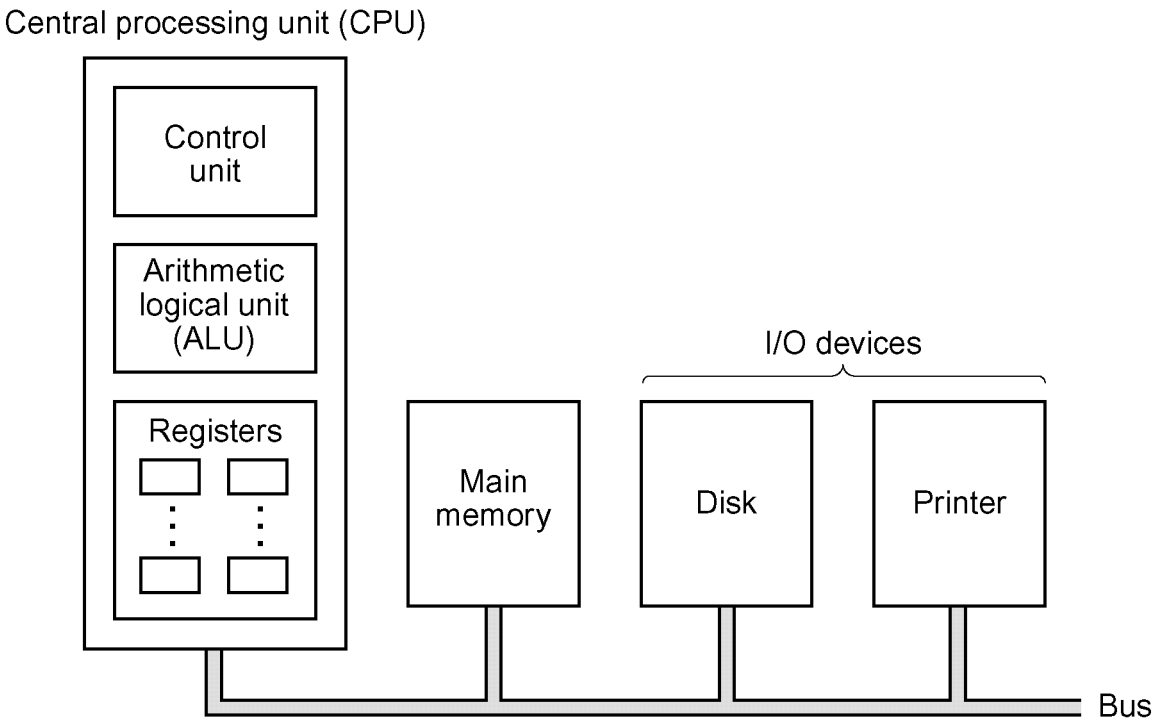

Figure 2-1. The organization of a simple computer with one CPU and two I/O devices.

2.1.1 Organisation du CPU

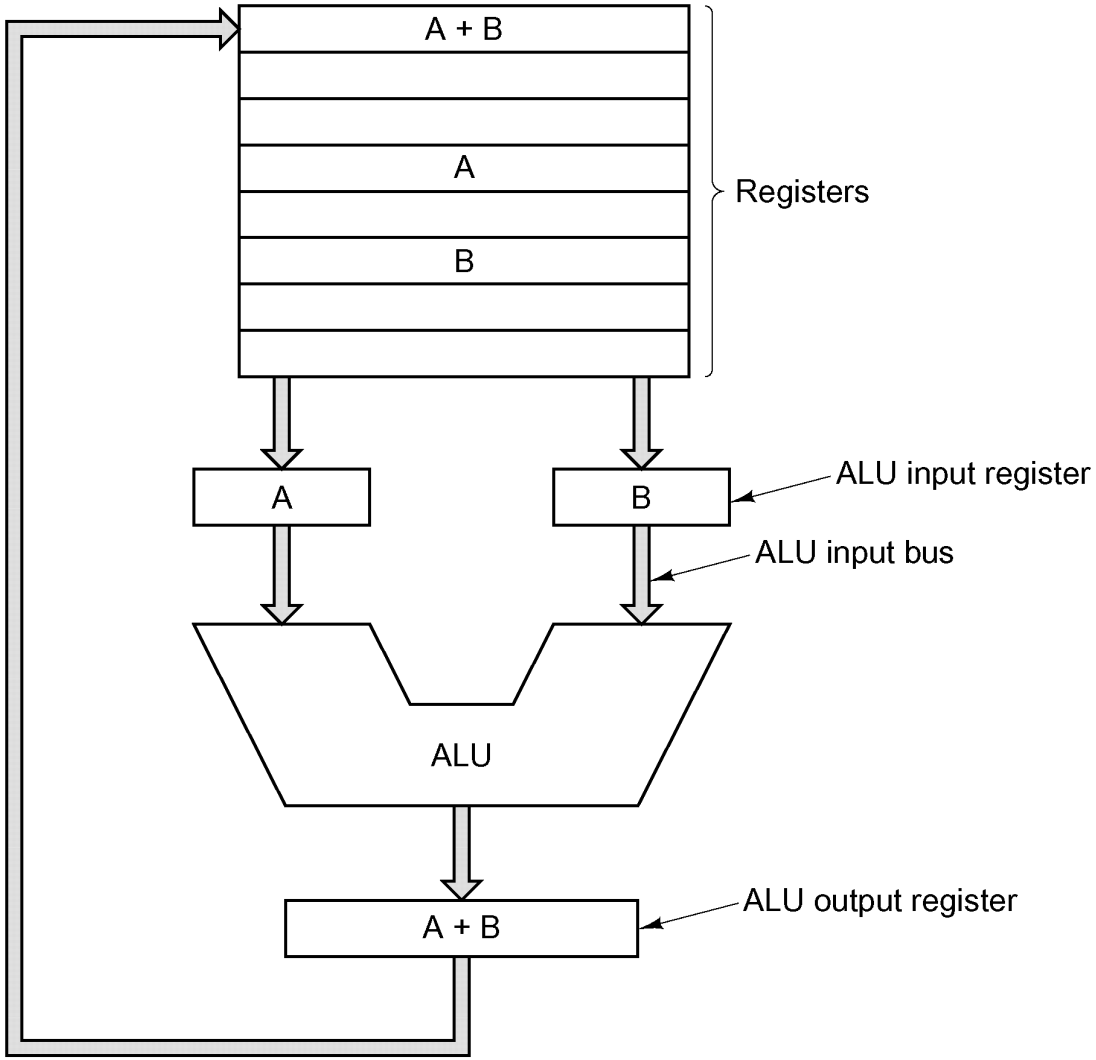

Figure 2-2. The data path of a typical von Neumann machine.

2.1.1 Organisation du CPU

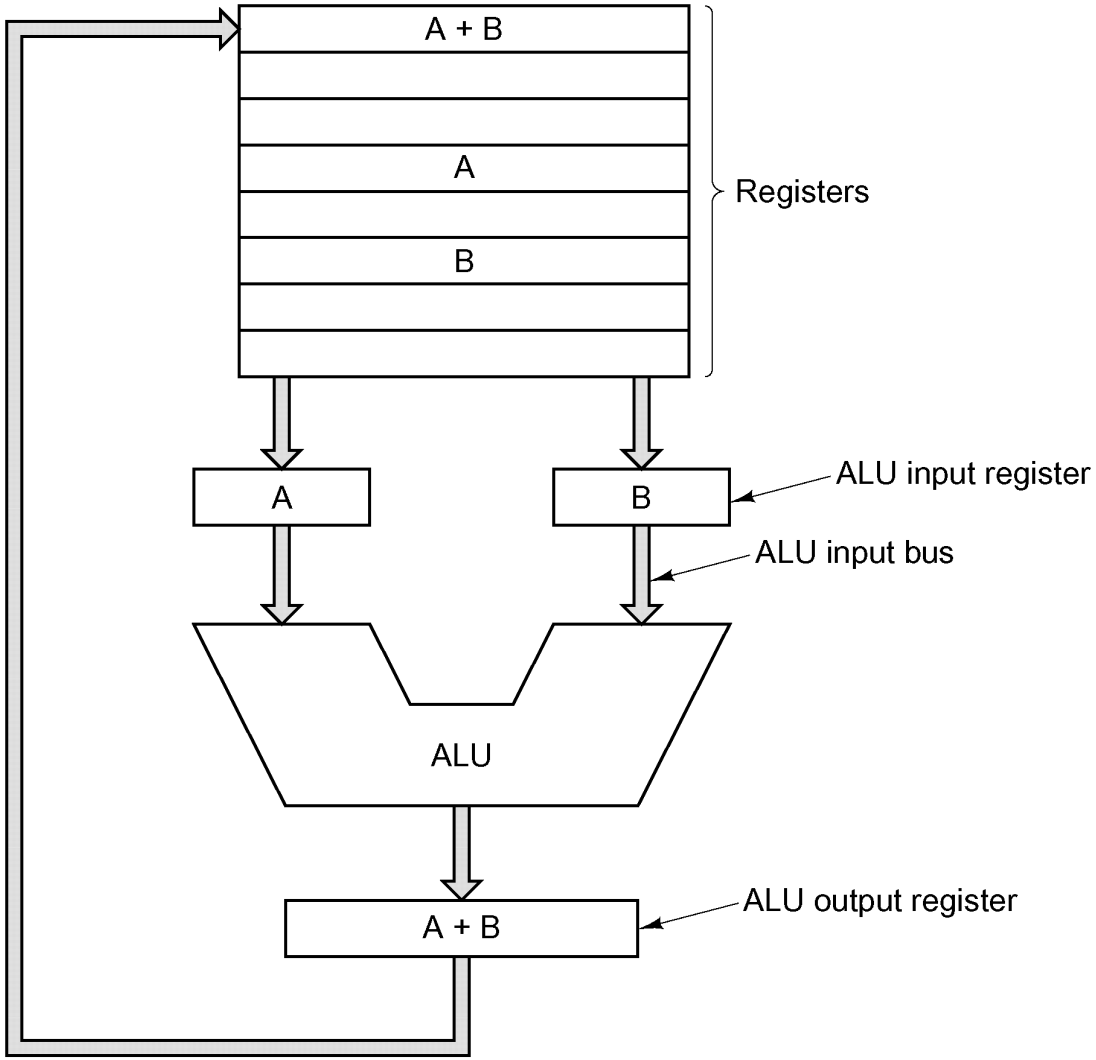

Figure 2-2. The data path of a typical von Neumann machine.

2.1.2 Exécution d'une instruction (1/2)

1. Charger la prochaine instruction à exécuter depuis la mémoire dans le registre instruction

2. Modifier le compteur ordinal pour qu'il pointe sur la prochaine instruction

3. Décoder l'instruction que l'on vient de charger

4. Localiser en mémoire d'éventuelles données nécessaires à l'instruction

5. Charger, si nécessaire, les données dans les registres généraux

6. Exécuter l'instruction

7. Revenir à l'étape 1

2.1.2 Exécution d'une instruction (2/2)

```
public class Interp {
   static int PC;                        // program counter holds address of next instr
   static int AC;                        // the accumulator, a register for doing arithmetic
   static int instr;                     // a holding register for the current instruction
   static int instr_type;                // the instruction type (opcode)
   static int data_loc;                  // the address of the data, or –1 if none
   static int data;                      // holds the current operand
   static boolean run_bit = true;        // a bit that can be turned off to halt the machine

   public static void interpret(int memory[ ], int starting_address) {
      // This procedure interprets programs for a simple machine with instructions having
      // one memory operand. The machine has a register AC (accumulator), used for
      // arithmetic. The ADD instruction adds am integer in memory to the AC, for example
      // The interpreter keeps running until the run bit is turned off by the HALT instruction.
      // The state of a process running on this machine consists of the memory, the
      // program counter, the run bit, and the AC. The input parameters consist of
      // of the memory image and the starting address.

      PC = starting_address;
      while (run_bit) {
         instr = memory[PC];             // fetch next instruction into instr
         PC = PC + 1;                    // increment program counter
         instr_type = get_instr_type(instr);          // determine instruction type
         data_loc = find_data(instr, instr_type);     // locate data (–1 if none)
         if (data_loc >= 0)              // if data_loc is –1, there is no operand
            data = memory[data_loc];     // fetch the data
         execute(instr_type, data);      //execute instruction
      }

   }

   private static int get_instr_type(int addr) { ... }
   private static int find_data(int instr, int type) { ... }
   private static void execute(int type, int data){ ... }
}
```

Figure 2-3. An interpreter for a simple computer (written in Java).

Instructions simples ou complexes et performances (1)

- 1ers ordinateurs: instructions simples et en nombre limité

Mais…

- Besoin de puissance et de performance conduisit à des instructions de plus en plus complexes et nombreuses:
  par exemple instructions arithmétiques « en virgule flottante » (manipulation de nombres « réels »), manipulation de chaînes de caractères, de tableaux (« vecteurs », « matrices »,…)

  instructions complexes plus efficaces que des suites d'instructions simples équivalentes, car possibilité de les exécuter en parallèle grâce à quelques composants additionnels (circuits), moyennant un certain coût supplémentaire du matériel.

Idée de « famille d'ordinateurs » (IBM): compatibilité de langages sur des machines différentes, plus ou moins efficaces
= Origine de la notion d' « architecture » (une même architecture pour différentes implémentations matérielles).
Possibilité d'exécuter un même programme sur différentes machines, efficacité selon prix.

Instructions simples ou complexes et performances(2)

Idée d' « interprétation »:

Permet de concevoir des ordinateurs simples et bon marché, mais permettant d'exécuter des instructions complexes

Exemple: gamme IBM 360, ordinateurs compatibles entre eux, mais large gamme de prix et performances (ceux qui interprètent les instructions complexes directement par le matériel étant les plus chères)

Avantages:

- Indépendance entre définition d'instructions complexes et implémentation
- Ajout possible d'instructions complexes après coup avec faible coût
- Structuration efficace du développement, test et documentaion des instructions complexes.
- Masquage possible de certains défauts du matériel

Facteur d'encouragement:

- Arrivée des ROM (Read Only Memory) pour réaliser des « mémoires de commande » contenant les instructions de l'interpréteur (micro-instructions). Accès rapide et relativement bon marché.

2.1.3 RISC versus CISC

Jeux d'instructions réduites vs complexes

- RISC = Reduced Instruction Set Computer
- CISC = Complex Instruction Set Computer

Tendance jusqu'en 1980: CISC

- IBM, Intel, Motorola, …

Dès 1980: RISC (initié par IBM !!!)

- John Coke (IBM)
- David Petterson und Carlo Séquin (Berkeley)
- Hennessy (Stanford): MIPS und SPARC (Sun !)

Aujourd'hui

- CISC: Pentium (Intel), …
- RISC: PowerPC (Motorola, IBM, Apple), Alpha (DEC), …

2.1.4 Principes de conception des ordinateurs

- Toute instruction est traitée directement par des composants matériels
- Maximiser la vitesse d'exécution des instructions
  - MIPS = Million Instruction Per Second (!)
  - GIPS = Billion Instruction Per Second (!)
- Les instructions doivent être simples à décoder
- Seules les instructions de rangement et de chargement doivent faire référence à la mémoire principale
- Il faut disposer d'un nombre important de registres

2.1.4 Principes de conception des ordinateurs

- Toute instruction est traitée directement par des composants matériels
- Maximiser la vitesse d'exécution des instructions
  - MIPS = Million Instruction Per Second (!)
  - GIPS = Billion Instruction Per Second (!)
- Les instructions doivent être simples à décoder
- Seules les instructions de rangement et de chargement doivent faire référence à la mémoire principale
- Il faut disposer d'un nombre important de registres

2.1.5 Parallélisme des instructions

a) Technique du pipeline ou pipelining

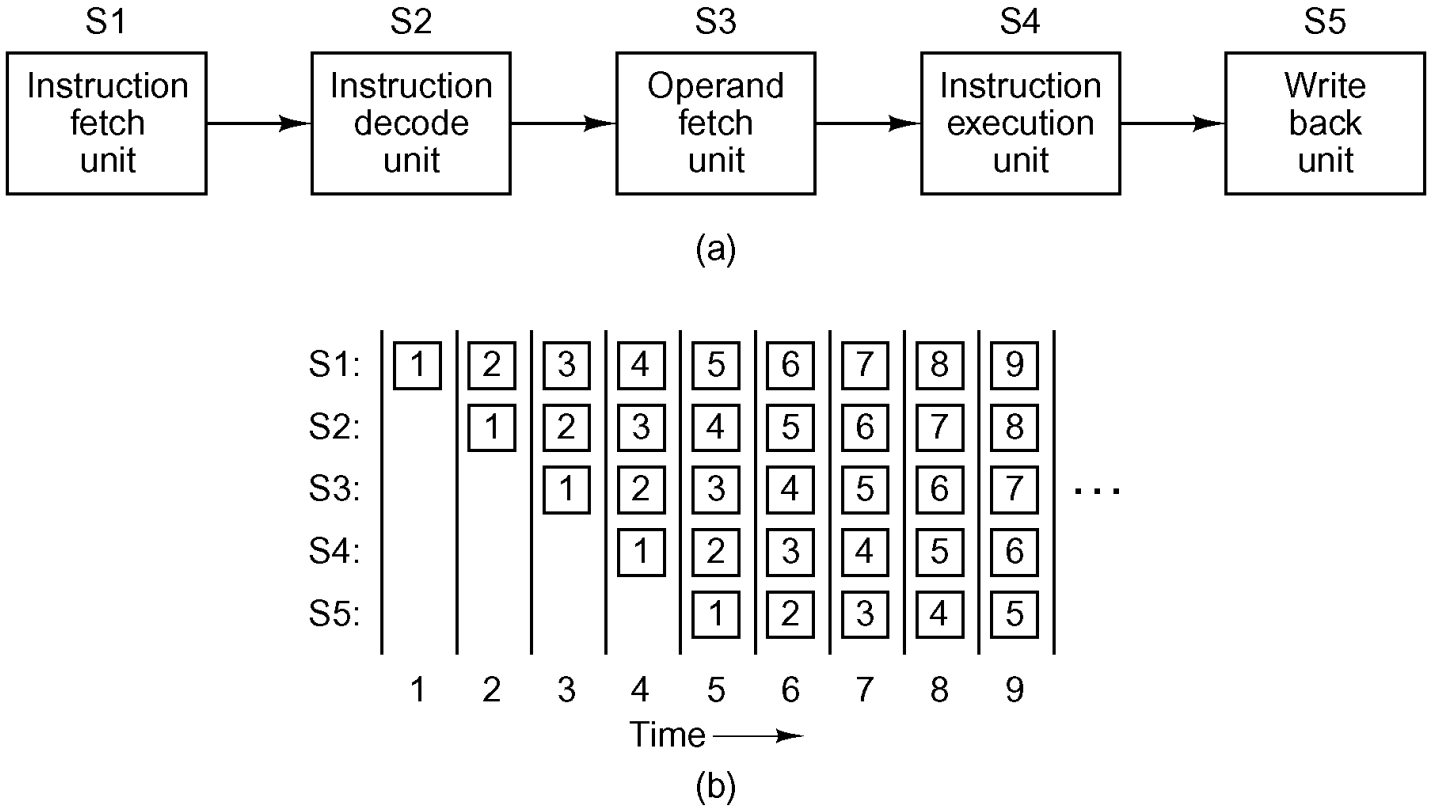

Figure 2-4. (a) A five-stage pipeline. (b) The state of each stage as a function of time. Nine clock cycles are illustrated.

(a) : Temps de latence (= temps d'exécution d'une instruction) vs
(b) : Bande passante du processeur (= nombre de MIPS)

2.1.5 Parallélisme des instructions

b) Architecture superscalaire (1/2)

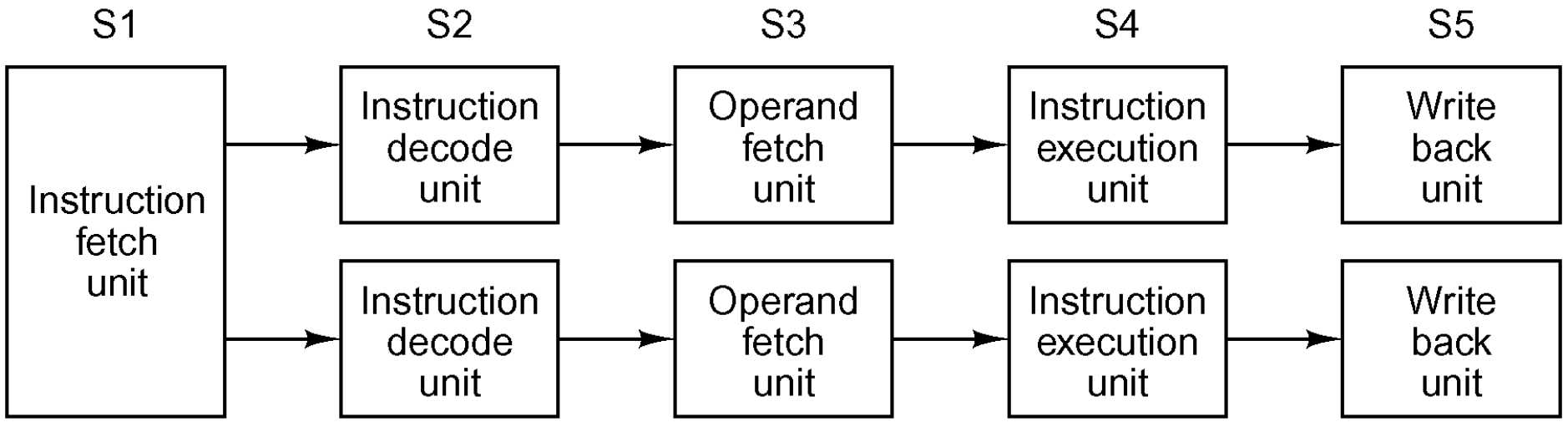

Figure 2-5. (a) Dual five-stage pipelines with a common instruction fetch unit.

2.1.5 Parallélisme des instructions

b) Architecture superscalaire (2/2)

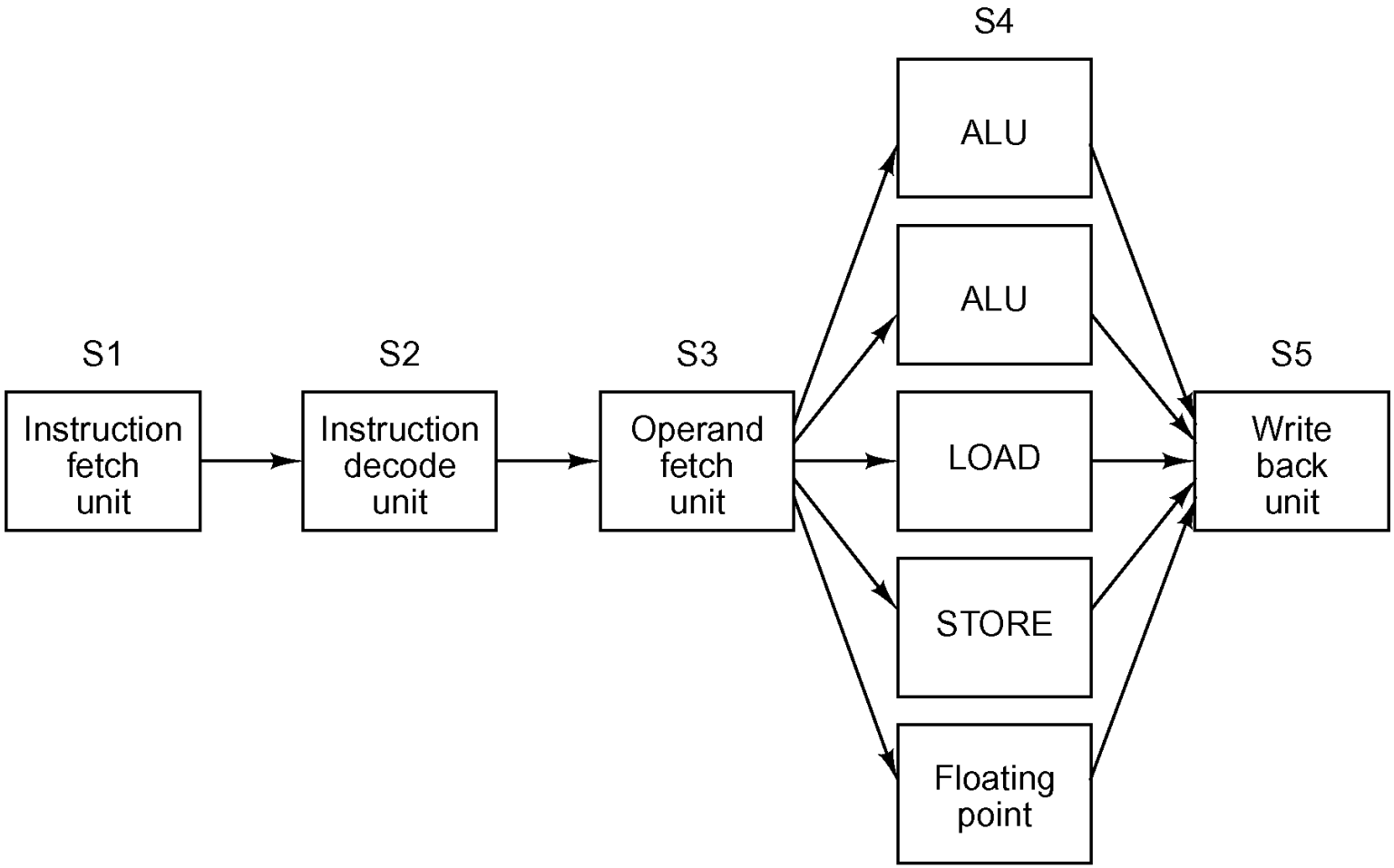

Figure 2-6. A superscalar processor with five functional units.

2.1.6 Parallélisme du processeur

Processeurs matriciel et vectoriel

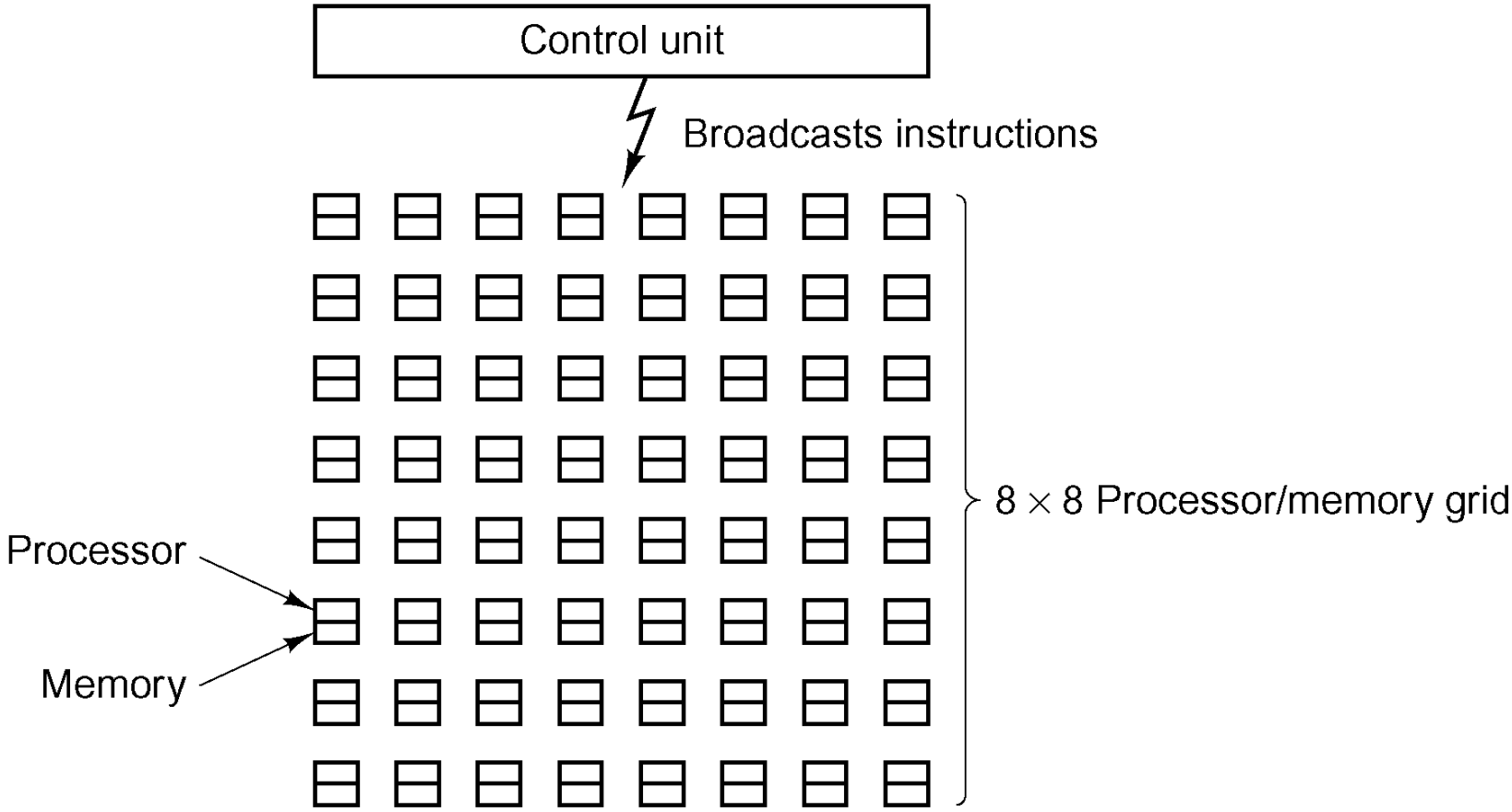

Figure 2-7. An array processor of the ILLIAC IV type.

2.1.6 Parallélisme du processeur

Multiprocesseurs sans/avec mémoire locale

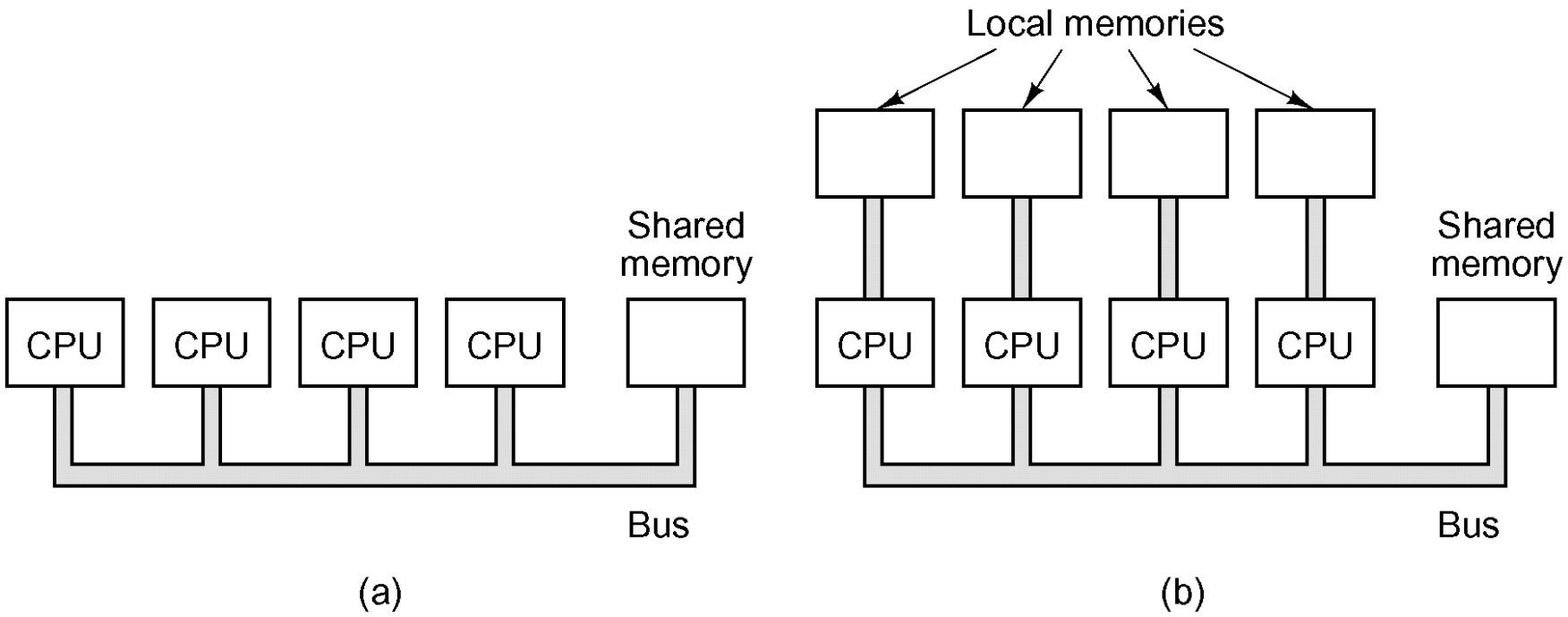

Figure 2-8. (a) A single-bus multiprocessor. (b) A multicomputer with local memories.

2.1.6 Parallélisme du processeur
Multi-ordinateurs

Une autre voie pour améliorer les performances des ordinateurs

Permet de dépasser les limites atteintes par les autres formes de parallélisme due à la complexité obtenue lorsque le nombre de processeurs devient trop grand

Systèmes distribués (ou « multi-ordinateurs »)

On connecte des ordinateurs par un réseau.

Topologies variées: anneaux, arborescences, grilles 2D ou 3D,…

Ils communiquent par échange de messages:
parfois on met en place des mécanismes permettant de « traverser » rapidement un ordinateur (nœud)

Connexion jusqu'à 10'000 ordinateurs déjà réalisée

Autres voies:

GRID et utilisation de réseaux standards (non dédiés) comme machine distribuée