# *CS1101: Lecture 14*
# Computer Systems Organization: Processors & Parallelism

Dr. Barry O'Sullivan
`b.osullivan@cs.ucc.ie`

Course Homepage
`http://www.cs.ucc.ie/~osullb/cs1101`

## Lecture Outline

- Design Principles for Modern Computers
- Parallelism
- Instruction-Level Parallelism
  - Pipelining
  - Dual Pipelines
  - Superscalar Architectures
- Processor-Level Parallelism
  - Array Computers
  - Multiprocessors
  - Multicomputers
- **Reading**: Tanenbaum, Chapter 2 Section 1

## Design Principles for Modern Computers

There is a set of design principles, sometimes called the RISC design principles, that architects of general-purpose CPUs do their best to follow:

- All Instructions Are Directly Executed by Hardware
  - eliminates a level of interpretation
- Maximise the Rate at Which Instructions are Issued
  - **MIPS** = millions of instructions per second
  - MIPS speed related to the number of instructions issued per second
  - Parallelism can play a role

## Design Principles for Modern Computers

- Instructions Should be Easy to Decode
  - a critical limit on the rate of issue of instructions
  - make instructions regular, fixed length, with a small number of fields.
  - the fewer different formats for instructions. the better.
- Only Loads and Stores Should Reference Memory
  - operands for most instructions should come from- and return to- registers.
  - access to memory can take a long time
  - thus, only LOAD and STORE instructions should reference memory.
- Provide Plenty of Registers
  - accessing memory is relatively slow, many registers (at least 32) need to be provided, so that once a word is fetched, it can be kept in a register until it is no longer needed.

## Parallelism

- Computer architects are constantly striving to improve performance of the machines they design.
- Making the chips run faster by increasing their clock speed is one way,
- However, most computer architects look to parallelism (doing two or more things at once) as a way to get even more performance for a given clock speed.
- Parallelism comes in two general forms:
  - instruction-level parallelism, and
  - processor-level parallelism.

## Instruction-Level Parallelism

- Parallelism is exploited within individual instructions to get more instructions/sec out of the machine.
- We will consider two approached
  - Pipelining
  - Superscalar Architectures

## Pipelining

- Fetching of instructions from memory is a major bottleneck in instruction execution speed. However, computers have the ability to fetch instructions from memory in advance
- These instructions were stored in a set of registers called the **prefetch buffer**.
- Thus, instruction execution is divided into two parts: fetching and actual execution;
- The concept of a **pipeline** carries this strategy much further.
- Instead of dividing instruction execution into only two parts, it is often divided into many parts, each one handled by a dedicated piece of hardware, all of which can run in parallel.

## A Example of Pipelining

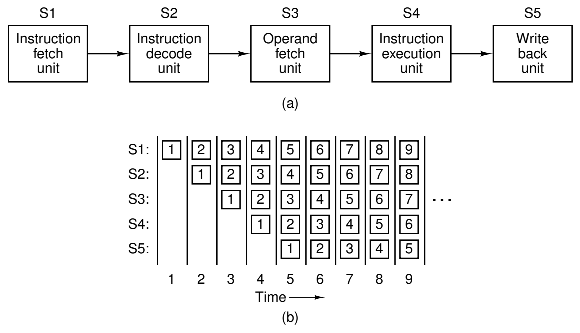

**Figure 2-4.** (a) A five-stage pipeline. (b) The state of each stage as a function of time. Nine clock cycles are illustrated.

## Dual Pipelines

- If one pipeline is good, then surely two pipelines are better.
- Here a single instruction fetch unit fetches pairs of instructions together and puts each one into its own pipeline, complete with its own ALU for parallel operation.
- To be able to run in parallel, the two instructions must not conflict over resource usage (e.g., registers), and neither must depend on the result of the other.

## Example: Dual Pipelines

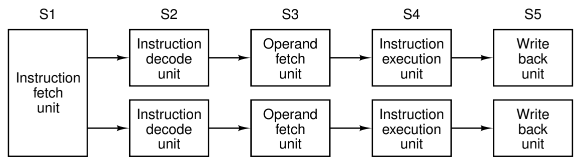

**Figure 2-5.** (a) Dual five-stage pipelines with a common instruction fetch unit.

## Superscalar Architectures

- Going to four pipelines is conceivable, but doing so duplicates too much hardware
- Instead, a different approach is used on high-end CPUs.
- The basic idea is to have just a single pipeline but give it multiple functional units.
- This is a **superscalar architecture** – using more than one ALU, so that more than one instruction can be executed in parallel.
- Implicit in the idea of a superscalar processor is that the S3 stage can issue instructions considerably faster than the S4 stage is able to execute them.

## Superscalar Architectures

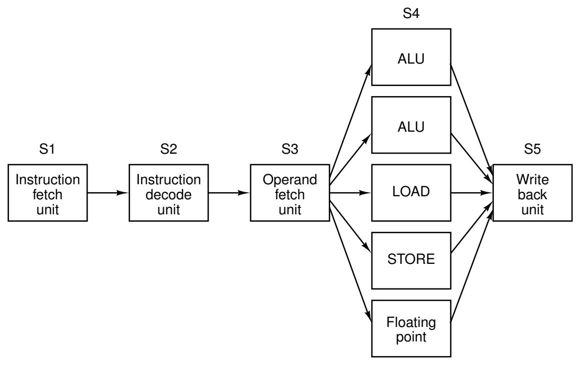

**Figure 2-6.** A superscalar processor with five functional units.

## Processor-Level Parallelism

- Instruction-level parallelism (pipelining and superscalar operation) rarely win more than a factor of five or ten in processor speed.
- To get gains of 50, 100, or more, the only way is to design computers with multiple CPUS
- We will consider three alternative architectures:
  - Array Computers
  - Multiprocessors
  - Multicomputers

## Array Computers

- An **array processor** consists of a large number of identical processors that perform the same sequence of instructions on different sets of data.
- A **vector processor** is efficient at at executing a sequence of operations on pairs of Data elements; all of the addition operations are performed in a single, heavily-pipelined adder.

## Example: Array Computers

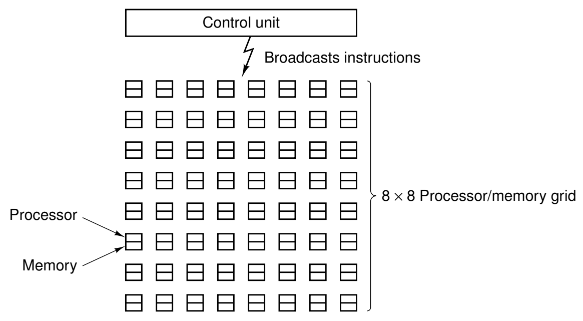

**Figure 2-7.** An array processor of the ILLIAC IV type.

## Multiprocessors

- The processing elements in an array processor are not independent CPUS, since there is only one control unit.
- The first parallel system with multiple full-blown CPUs is the **multiprocessor**.
- This is a system with more than one CPU sharing a common memory co-ordinated in software.
- The simplest one is to have a single bus with multiple CPUs and one memory all plugged into it.

## Example: Multiprocessors

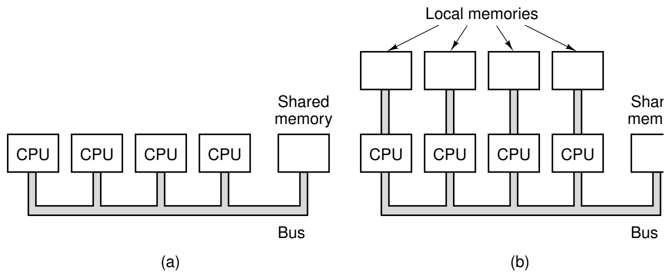

**Figure 2-8.** (a) A single-bus multiprocessor. (b) A multicomputer with local memories.

## Multicomputers

- Although multiprocessors with a small number of processors ($< 64$) are relatively easy to build, large ones are surprisingly difficult to construct.
- The difficulty is in connecting all the processors to the memory.
- To get around these problems, many designers have simply abandoned the idea of having a shared memory and just build systems consisting of large numbers of interconnected computers, each having its own private memory, but no common memory.
- These systems are called **multicomputers**.