Review of

Computer Organization

Operating System (CSC1036 & INF1036)

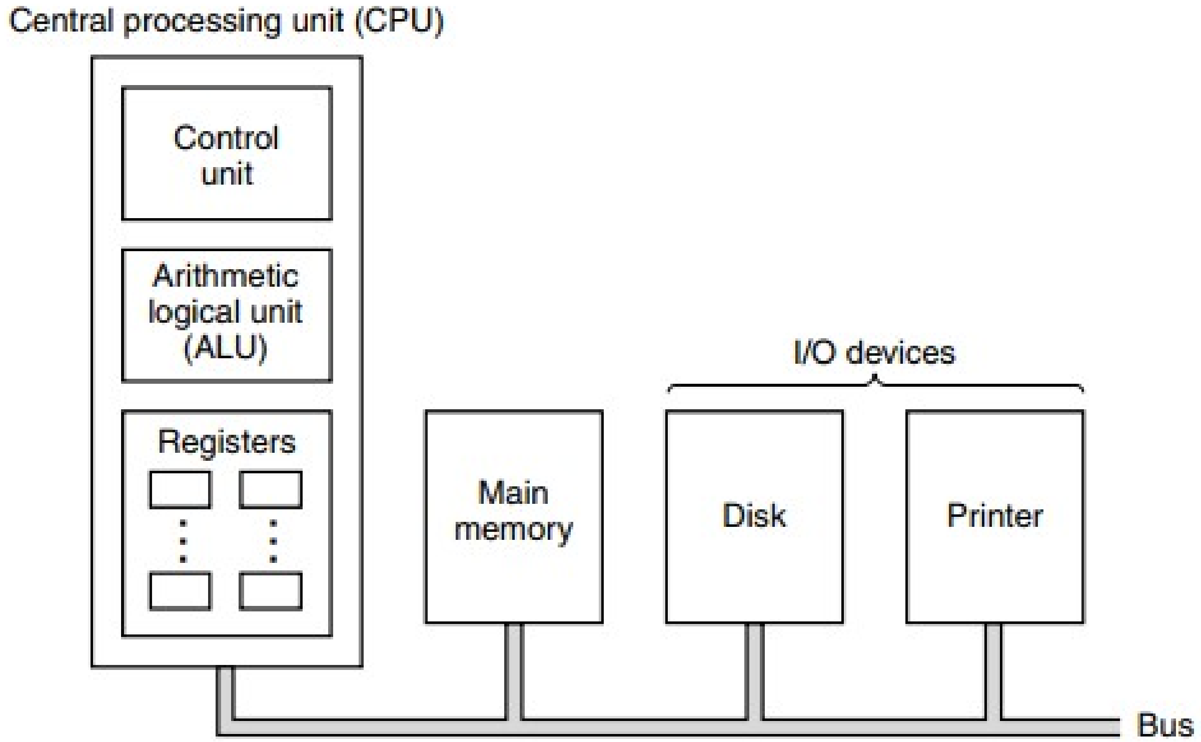
Dr. Hasin A. Ahmed
Assistant Professor
Department of Computer Science
Gauhati University

PROCESSORS

CPU

- The CPU (Central Processing Unit) is the “brain” of the computer
- Components are connected by a bus, which is a collection of parallel wires for transmitting address, data, and control signals
- Buses can be external to the CPU, connecting it to memory and I/O devices, but also internal to the CPU

CPU


Central processing unit (CPU)
Control unit
Arithmetic logical unit (ALU)
Registers
Main memory
I/O devices
Disk
Printer
Bus

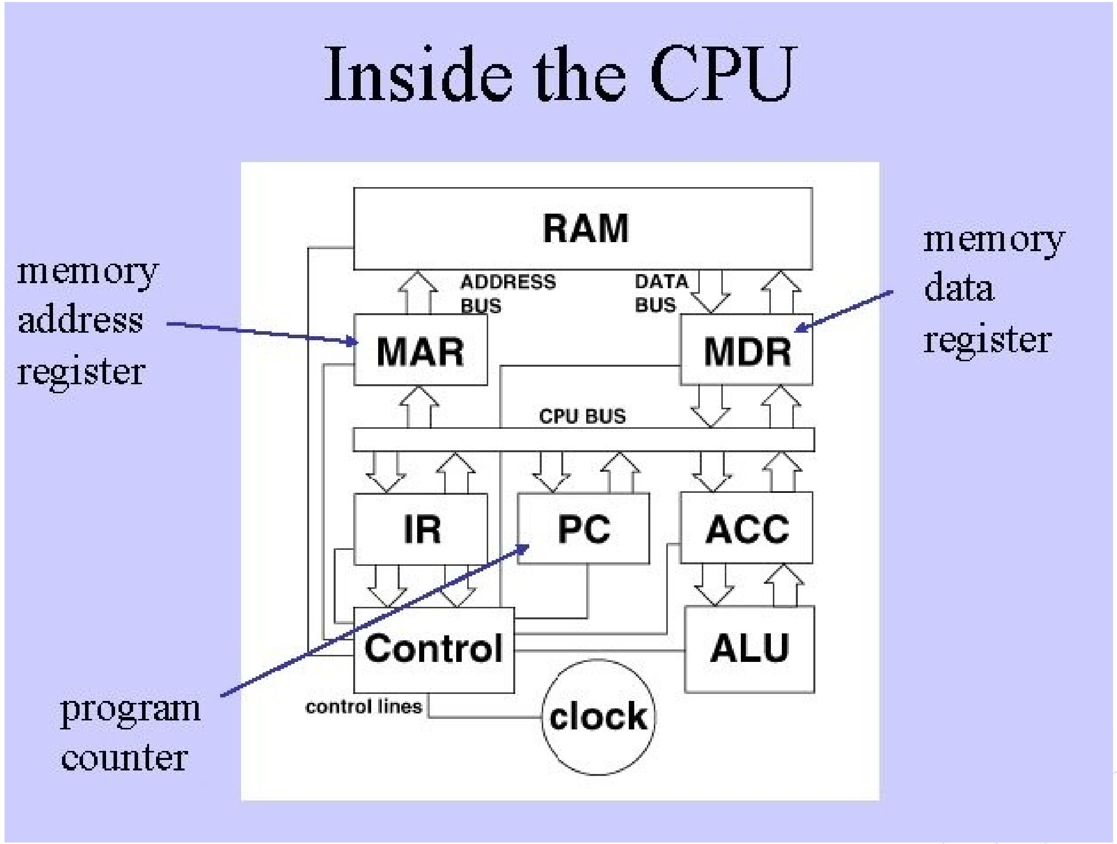
CPU

- The CPU also contains a small, high-speed memory used to store temporary results and certain control information
- This memory is made up of a number of registers
- Usually, all the registers have the same size.
- Registers can be read and written at very high speed

CPU

- Register types: General purpose & Special purpose
- The most important register is the Program Counter (PC) which points to the next instruction to be fetched for execution
- Also important is the Instruction Register (IR) which holds the instruction currently being executed

MAR and MDR


Inside the CPU
RAM
ADDRESS BUS
DATA BUS
MAR
MDR
CPU BUS
IR
PC
ACC
Control
ALU
control lines
clock
memory address register
memory data register
program counter

Register-memory and register-register instruction

- Most instructions can be divided into one of two categories: register-memory or register-register
- Register-memory instructions allow memory words to be fetched into registers or allow registers to be stored back into memory

Register-memory and register-register instruction

- A typical register-register instruction fetches two operands from the registers, brings them to the ALU input registers, performs some operation on them and stores the result back in one of the registers
- The process of running two operands through the ALU and storing the result is called the data path cycle

Fetch-decode-execute cycle

1) Fetch the next instruction from memory into the instruction register.
2) Change the program counter to point to the following instruction.
3) Determine the type of instruction just fetched.
4) If the instruction uses a word in memory, determine where it is.
5) Fetch the word, if needed, into a CPU register.
6) Execute the instruction.
7) Go to step 1 to begin executing the following instruction

Hardware and Software Implementation of Instructions

- Instead of CPU, interpreter can be made to execute instruction
- This equivalence between hardware processors and interpreters has important implications for computer organization
- After designing machine language for a new computer, the design team can decide whether they want to build a hardware processor or to write an interpreter to interpret programs
- If they choose to write an interpreter, they must also provide some hardware machine to run the interpreter

Hardware and Software Implementation of Instructions

- The more complex instructions were better because the execution of individual operations could sometimes be overlapped or otherwise executed in parallel using different hardware.
- For expensive, high-performance computers, the cost of this extra hardware could be readily justified.
- However, instruction compatibility requirements and the rising cost of software development created the need to implement complex instructions even on low-end computers where cost was more important than speed.

Hardware and Software Implementation of Instructions

- By the late 1950s, IBM had recognized that supporting a single family of machines, all of which executed the same instructions, had many advantages
- IBM introduced the term architecture to describe this level of compatibility
- But how to build a low-cost computer that could execute all the complicated instructions of high-performance, expensive machines?

Hardware and Software Implementation of Instructions

- The answer lay in interpretation.
- IBM System/360 architecture, a compatible family of computers used direct hardware only on the most expensive models.

RISC vs CISC

- In 1980, a group at Berkeley led by David Patterson and Carlo Séquin began designing VLSI CPU chips that did not use interpretation
- They coined the term RISC for this concept and named their CPU chip the RISC I CPU, followed shortly by the RISC II
- These new processors were significantly different than commercial processors of the day.
- Since they did not have to be backward compatible with existing products, their designers were free to choose new instruction sets that would maximize total system performance.

RISC vs CISC

- The characteristic of RISC that caught everyone’s attention was the relatively small number of instructions available, typically around 50.
- This number was far smaller than the 200 to 300 on established computers such as the DEC VAX and the large IBM mainframes.
- In fact, the acronym RISC stands for Reduced Instruction Set Computer, which was contrasted with CISC, which stands for Complex Instruction Set Computer

RISC vs CISC

- A great religious war ensued, with the RISC supporters attacking the established order (VAX, Intel, large IBM mainframes).
- They claimed that the best way to design a computer was to have a small number of simple instructions that execute in one cycle of the data path
- Their argument was that even if a RISC machine takes four or five instructions to do what a CISC machine does in one instruction, if the RISC instructions are 10 times as fast, RISC wins

RISC vs CISC

- One might think that given the performance advantages of RISC technology, RISC machines (such as the Sun UltraSPARC) would have mowed down CISC machines (such as the Intel Pentium) in the marketplace.
- Nothing of that short happened

RISC vs CISC

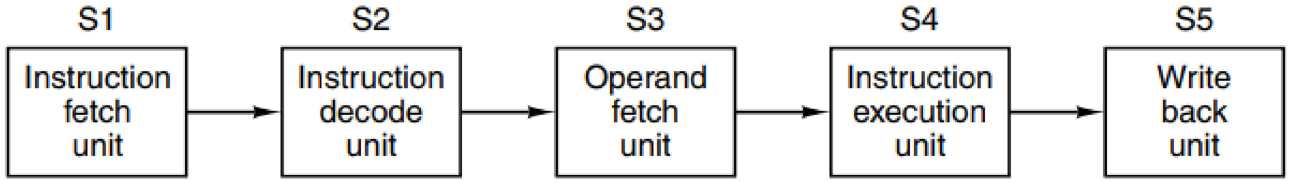
- There is the issue of backward compatibility and the billions of dollars companies have invested in software for the Intel line
- Intel has been able to employ the same ideas even in a CISC architecture. Starting with the 486, the Intel CPUs contain a RISC core that executes the simplest (and typically most common) instructions in a single data path cycle, while interpreting the more complicated instructions in the usual CISC way
- While this hybrid approach is not as fast as a pure RISC design, it gives competitive overall performance while still allowing old software to run unmodified.

RISC vs CISC

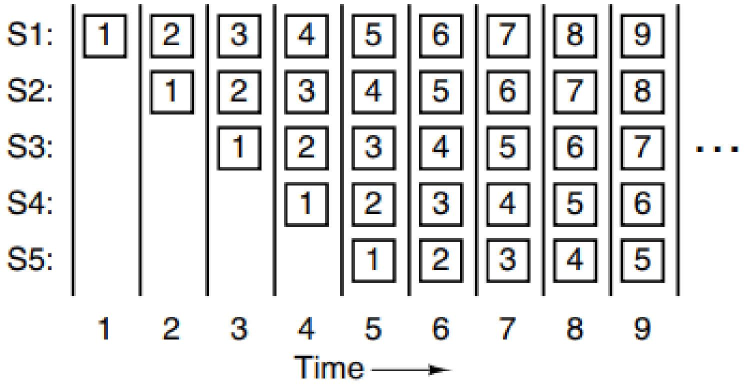
- This is a set of design principles, sometimes called the RISC design principles, that architects of new general-purpose CPUs do their best to follow
- All Instructions Are Directly Executed by Hardware
- Maximize the Rate at Which Instructions Are Issued
- Instructions Should Be Easy to Decode
- Only Loads and Stores Should Reference Memory
- Provide Plenty of Registers

Parallelism

- Parallelism comes in two general forms, namely, instruction-level parallelism and processor-level parallelism.


S1
Instruction fetch unit
S2
Instruction decode unit
S3
Operand fetch unit
S4
Instruction execution unit
S5
Write back unit

(a)


S1: 1 2 3 4 5 6 7 8 9
S2: 1 2 3 4 5 6 7 8
S3: 1 2 3 4 5 6 7
S4: 1 2 3 4 5 6
S5: 1 2 3 4 5
1 2 3 4 5 6 7 8 9
Time

(b)

Towards Superscalar Architecture

- If one pipeline is good, then surely two pipelines are better.
- One possible design for a dual pipeline CPU:

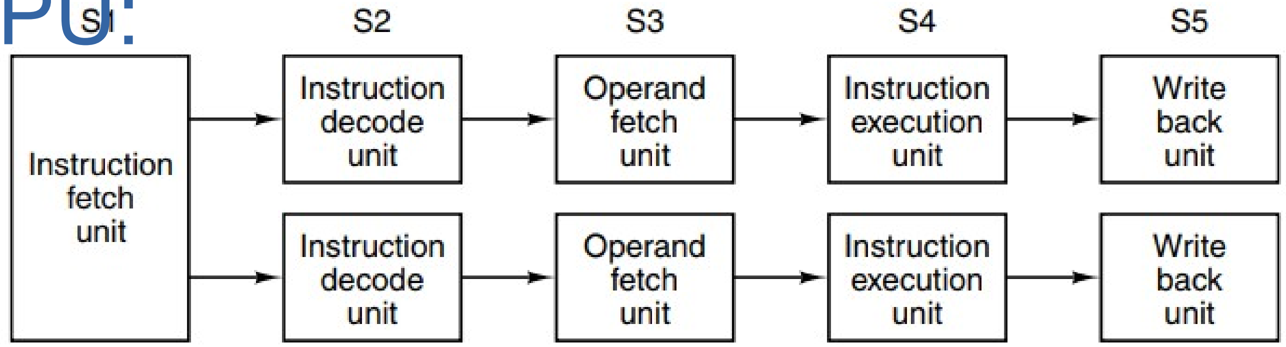


Figure 2-5. Dual five-stage pipelines with a common instruction fetch unit.

Resolving conflicts

- 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.
- As with a single pipeline, either the compiler must guarantee this situation to hold (i.e., the hardware does not check and gives incorrect results if the instructions are not compatible), or conflicts must be detected and eliminated during execution using extra hardware.

Superscalar Architecture

- Going to four pipelines is conceivable, but doing so duplicates too much hardware
- Instead, a different approach is used on high-end CPUs called Super scalar architecture that has just a single pipeline but uses multiple functional units

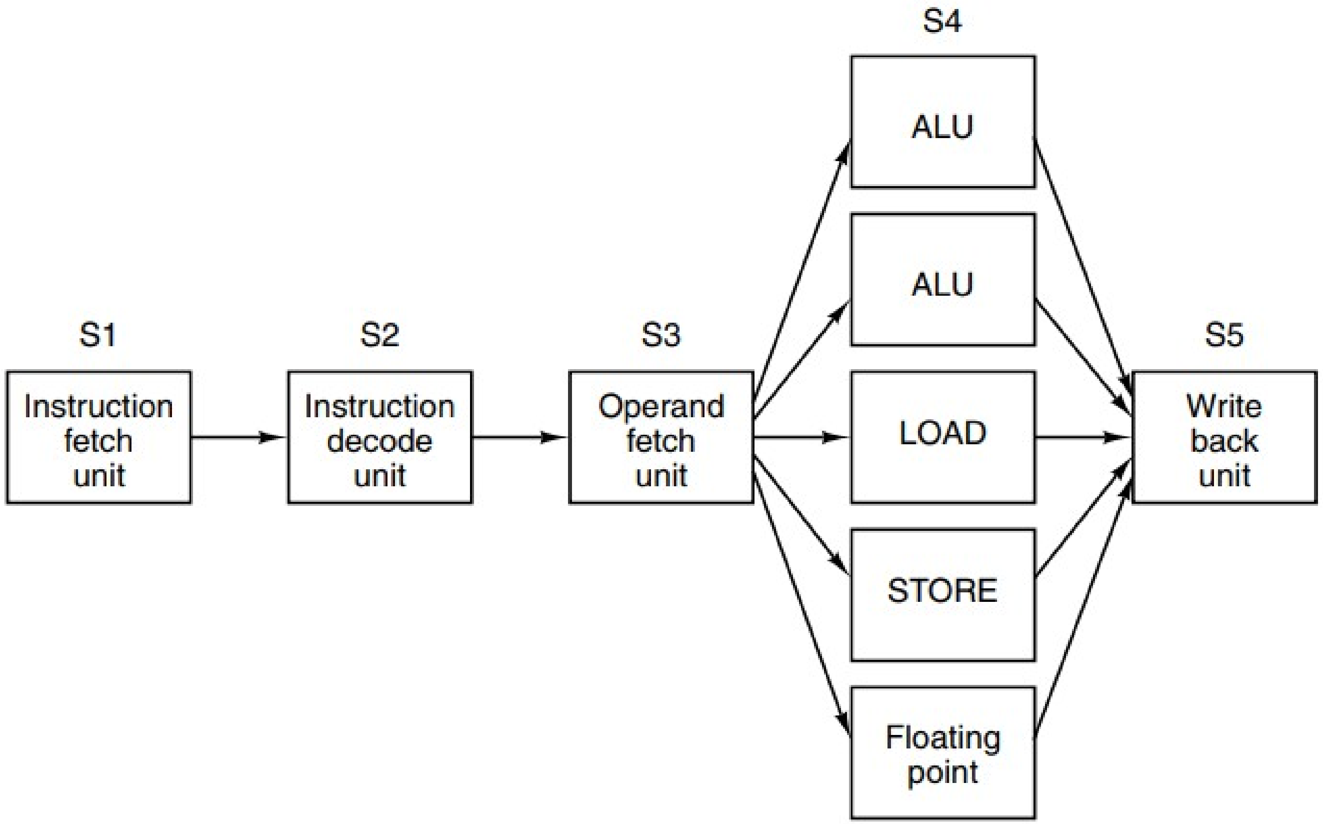


Figure 2-6. A superscalar processor with five functional units.

Data parallelism

- A substantial number of problems in computational domains such as the physical sciences, engineering, and computer graphics involve loops and arrays, or otherwise have a highly regular structure.
- Often the same calculations are performed repeatedly on many different sets of data.
- Two primary methods have been used to execute these highly regular programs quickly and efficiently: SIMD processors and vector processors.

Processor level parallelism

- A Single Instruction-stream Multiple Data-stream or SIMD processor consists of a large number of identical processors that perform the same sequence of instructions on different sets of data
- Modern graphics processing units (GPUs) heavily rely on SIMD processing to provide massive computational power

Vector Processor

- A vector processor appears to the programmer very much like a SIMD processor.
- Like a SIMD processor, it is very efficient at executing a sequence of operations on pairs of data elements.
- But unlike a SIMD processor, all of the operations are performed in a single, heavily pipelined functional unit.
- the vector processor has the concept of a vector register which consists of a set of conventional registers that can be loaded from memory in a single instruction and fed to the pipelined functional unit (e.g. adder)

Multiprocessor

- The processing elements in a data parallel processor are not independent CPUs, since there is only one control unit shared among all of them.
- Multiprocessor, a system with more than one CPU sharing a common memory

Multiprocessor implementation schemes

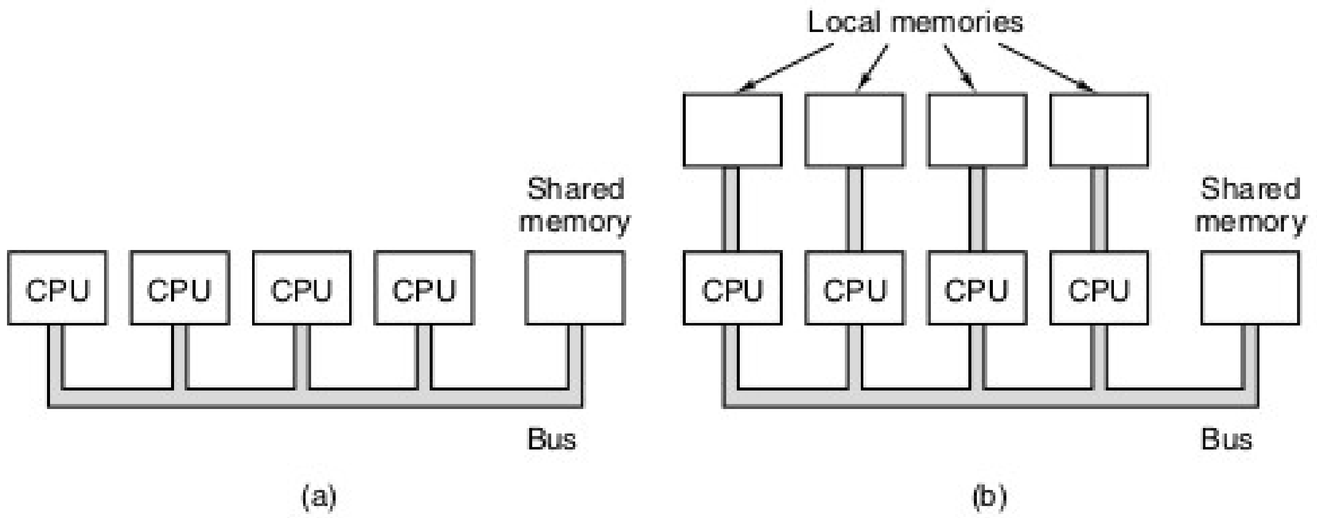


Figure 2-8. (a) A single-bus multiprocessor. (b) A multicomputer with local memories.

Multicomputer

- Although multiprocessors with a modest number of processors (≤ 256) are relatively easy to build, large ones are surprisingly difficult to construct.
- The difficulty is in connecting so many 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. The CPUs in a multicomputer are said to be loosely coupled, to contrast them with the tightly coupled multiprocessor CPUs.

thank you

? ? ?