System bus.

A computer system consists of three major components: a processor, a memory unit, and an input/output (I/O) subsystem. Interconnection network facilitates communication among these three components, as shown in Figure 21. The interconnection network is called the ***system bus***.

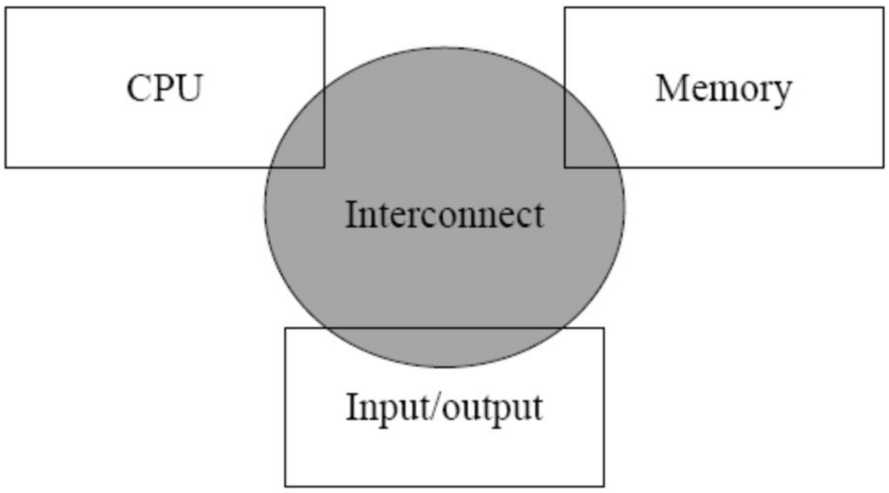


Figure 21 The three main components of a computer system are interconnected by a bus.

The term “bus” is used to represent a group of electrical signals or the wires that carry these signals. As shown in Figure 1.5, the system bus consists of three major components: an address bus, a data bus, and a control bus.

The address bus width determines the amount of physical memory addressable by the processor. The data bus width indicates the size of the data transferred between the processor and memory or an I/O device. For example, the Pentium processor has 32 address lines and 64 data lines. Thus, the Pentium can address up to 2^{32}, or 4 GB of memory. Furthermore, each data transfer can move 64 bits of data.

The control bus consists of a set of control signals. Typical control signals include memory read, memory write, I/O read, I/O write, interrupt, interrupt acknowledge, bus request, and bus grant. These control signals indicate the type of action taking place on the system bus. For example, when the processor is writing data into the memory, the memory write signal is generated. Similarly, when the processor is reading from an I/O device, it generates the I/O read signal.

A bus connects various components in a computer system. Thus buses can be categorize into:

Internal buses: The processor uses several internal buses to interconnect registers and the ALU.

External buses: are used to interface with the devices outside a typical processor system. By our classification, serial and parallel interfaces, Universal Serial Bus (USB). These buses are typically used to connect I/O devices.

Figure 22 shows *dedicated buses* connecting the major components of a computer system. The system bus consists of address, data, and control buses. One problem with the dedicated bus design is that it requires a large number of wires. We can reduce this count by using *multiplexed buses*. For example, a single bus may be used for both address and data. In addition, the address and data bus widths play an important role in determining the address space and data transfer rate, respectively.

• ***Bus Width****:* Bus width refers to the *data* and *address* bus widths. System performance improves with a wider data bus as we can move more bytes in parallel. We increase the addressing capacity of the system by adding more address lines.

• ***Bus Type:*** As discussed in the last section, there are two basic types of buses: *dedicated* and *multiplexed*.

• ***Bus Operations****:* Bus systems support several types of operations to transfer data. These include the *read, write, block transfer, read-modify-write,* and *interrupt.*

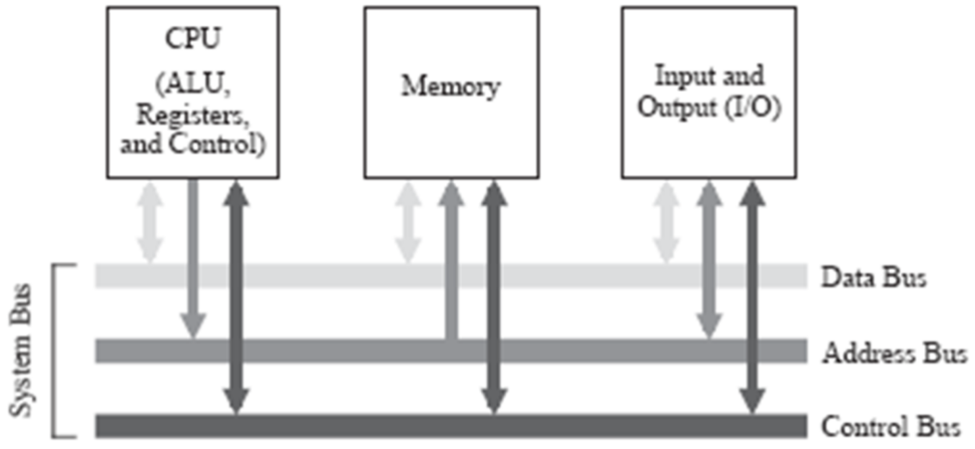


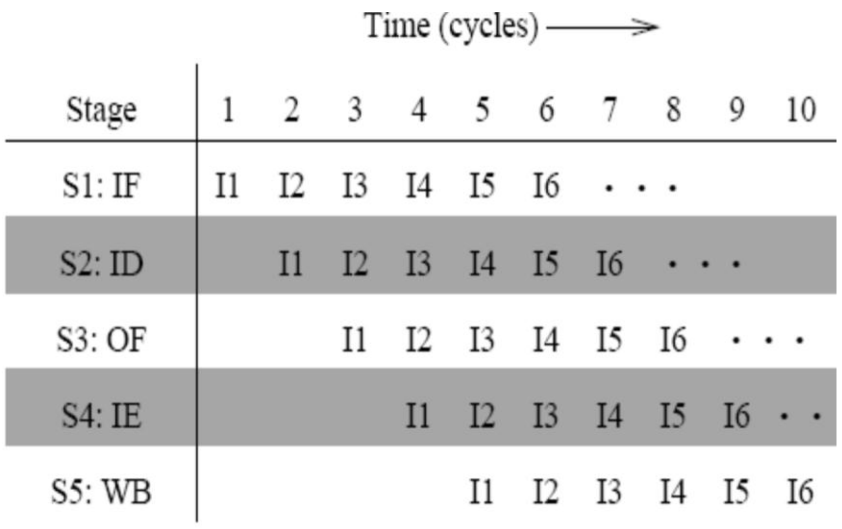
Figure 22: The system bus model of a computer system.

Consider, for example, the bus connecting a CPU and memory in a given system. The size of the memory in that system is **512M** word and each word is 32 bits. In such system, the size of the address bus should be $\log_2 (512 \times 2^{20}) = 29$ lines.

23- Pipelining.

In each execution cycle the control unit would have to wait until the instruction is fetched from memory. Furthermore, the ALU would have to wait until the required operands are fetched from memory. Processor speeds are increasing at a much faster rate than the improvements in memory speeds. Thus, we would be wasting the control unit and ALU resources by keeping them idle while the system fetches instructions and data. How can we avoid this situation? Let's suppose that we can prefetch the instruction. That is, we read the instruction before the control unit needs it. These prefetched instructions are typically placed in a set of registers called the *prefetch buffers*. Then, the control unit doesn't have to wait. How do we do this prefetch? Given that the program execution is sequential, we can prefetch the next instruction in sequence while the control unit is busy decoding the current instruction. Pipelining generalizes this concept of overlapped execution. Similarly, prefetching the required operands avoids the idle time experienced by the ALU.

Figure 23 shows how pipelining helps us improve the efficiency. The instruction execution can be divided into five parts. In pipelining terminology, each part is called a stage. For simplicity, let’s assume that execution of each stage takes the same time (say, one cycle). As shown in Figure 23, each stage spends one cycle in executing its part of the execution cycle and passes the instruction on to the next stage. Let’s trace the execution of this pipeline during the first few cycles. During the first cycle, the first stage S1 fetches the instruction. All other stages are idle. During Cycle 2, S1 passes the first instruction I1 to stage S2 for decoding and S1 initiates the next instruction fetch. Thus, during Cycle 2, two of the five stages are busy: S2 decodes I1 while S1 is busy with fetching I2. During Cycle 3, stage S2 passes instruction I1 to stage S3 to fetch any required operands. At the same time, S2 receives I2 from S1 for decoding while S1 fetches the third instruction. This process is repeated in each stage. As you can see, after four cycles, all five stages are busy. This state is called the pipeline full condition. From this point on, all five stages are busy.


Time (cycles) ⟶

Stage	1	2	3	4	5	6	7	8	9	10
S1: IF	I1	I2	I3	I4	I5	I6	• • •			
S2: ID		I1	I2	I3	I4	I5	I6	• • •		
S3: OF			I1	I2	I3	I4	I5	I6	• • •	
S4: IE				I1	I2	I3	I4	I5	I6	• •
S5: WB					I1	I2	I3	I4	I5	I6

Figure 23 A pipelined execution of the basic execution cycle

This figure clearly shows that the execution of instruction I1 is completed in Cycle 5. Thus, executing six instructions takes only 10 cycles. Without pipelining, it would have taken 30 cycles.

Notice from this description that pipelining does not speed up execution of individual instructions; each instruction still takes five cycles to execute. However, pipelining increases the number of instructions executed per unit time; that is, instruction throughput increases.

Pipelining a computer architecture designed so that all parts of circuit are always working, so that no part of the circuit is stalled waiting from another part. Pipelining allows overlapped execution to improve throughput.