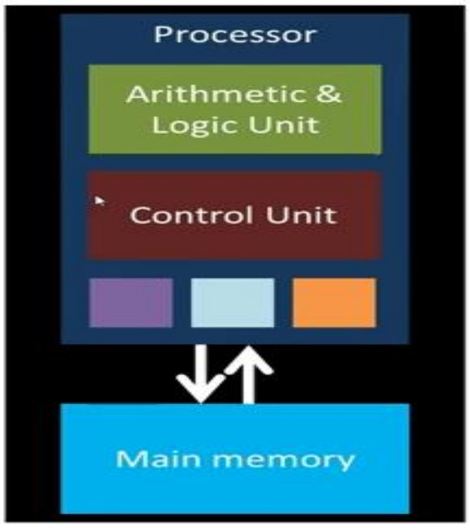
Contents

A digital computer contains four parts:

1. The CPU
2. memory to store data and instructions
3. the I/O devices
4. secondary storage



Components of a CPU

The CPU (microprocessor or processor) is the electronic circuit responsible for executing the instructions of a computer program and it contains:

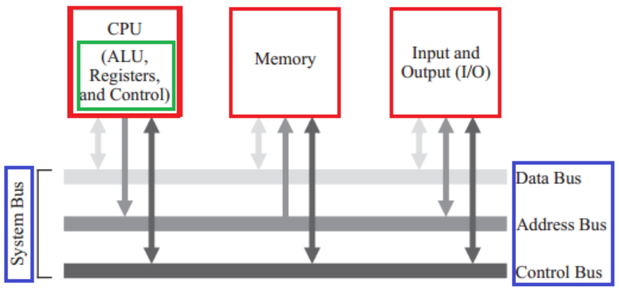
1. The **Arithmetic Logic Unit (ALU)**
2. The **Control Unit (CU)**
3. **Registers**

Registers are high speed storage areas in the CPU. ***All*** data must be stored in a register before it can be processed.

MAR	Memory Address Register	Holds the memory location of data that needs to be accessed
MDR	Memory Data Register	Holds data that is being transferred to or from memory
AC	Accumulator	Where intermediate arithmetic and logic results are stored
PC	Program Counter	Contains the address of the next instruction to be executed
CIR	Current Instruction Register	Contains the current instruction during processing

These three major components are interconnected together using the **System Bus** (electrical pathway), which is made up of:

- The Control Bus
- The Data Bus
- The Address Bus



- Numeric value on the **address bus** indicates where to go to in memory *or* which I/O device port locations, each of which has a unique binary numeric address.

Recall: memory address decoder functions

- Data is passed among CPU, I/O, and memory devices using the **data bus**.
- The **control bus** contains signals that determine the direction of the data transfer between memory and the I/O devices.

Following is a block diagram of a basic computer system: a top level view of the computer components

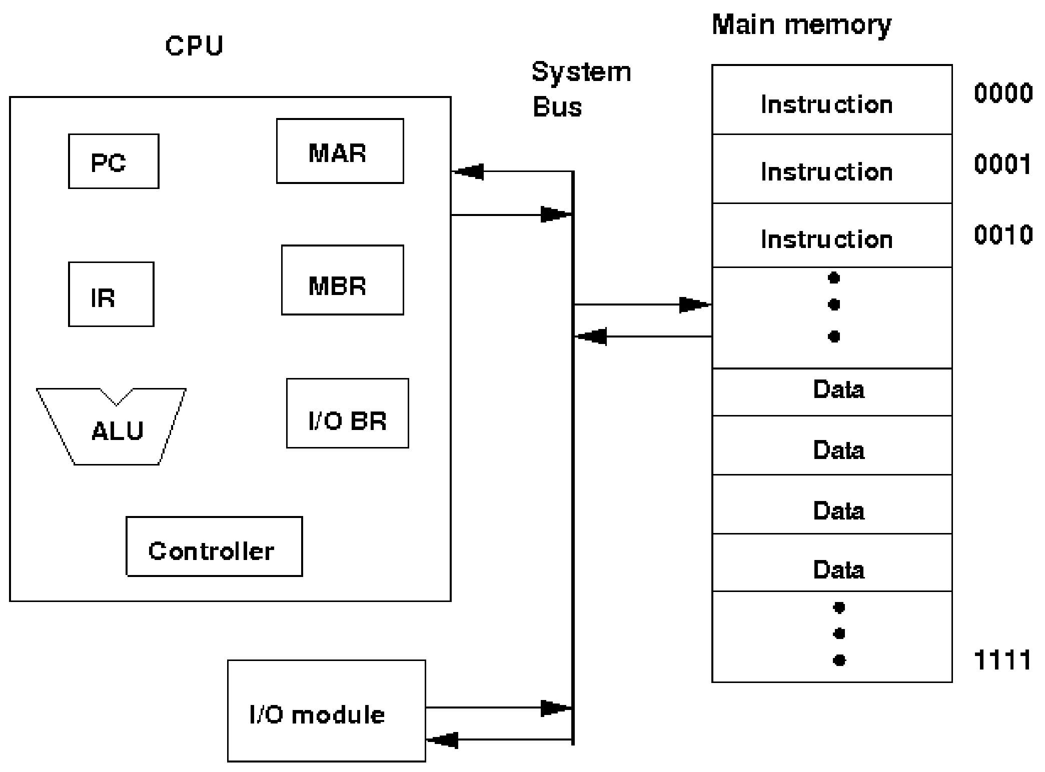


Figure 1 Block Diagram of a basic computer system

The CPU exchanges data with memory.

The memory unit consists of RAM, sometimes referred to as primary or main memory. Unlike a hard drive (secondary memory), this memory is fast and also directly accessible by the CPU.

RAM is split into partitions. Each partition consists of an address and its contents (both in binary form).

The address will uniquely identify every location in the memory.

Loading data from permanent memory (hard drive), into the faster and directly accessible temporary memory (RAM), allows the CPU to operate much quicker.

A processor typically makes use of internal (to the processor) registers:

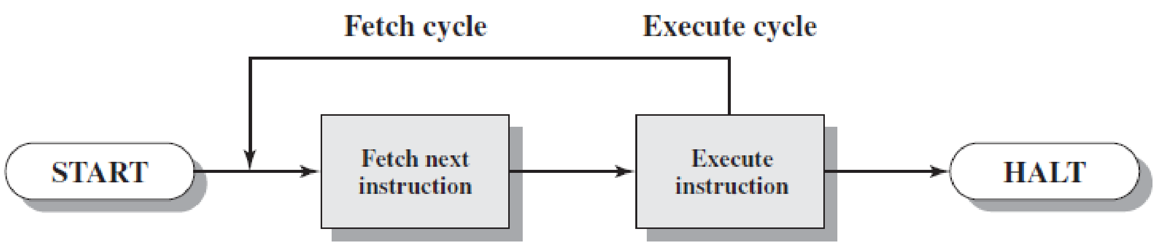
1. Memory address register (MAR) which specifies the address for the next read or write
2. Memory buffer register (MBR) which contains the data to be written into memory or receives the data read from memory
3. an I/O buffer (I/O BR) register is used for the exchange of data between an I/O module and the CPU
4. A memory module consists of a set of a locations defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data.
5. An I/O module transfers data from external devices to processor (CPU) and memory and vice versa. It contains internal buffers for temporarily holding data until they can be sent on.

Fetch Execute (and Decode) Cycle

The basic function of a computer is to execute a program which consists of a set of instructions stored in the memory. Processing required for a single instruction is called an instruction cycle which consists of an instruction **fetch** and instruction **execute**.

6. A register called program counter (PC) holds the address of the next instruction. Unless told otherwise the processor always increments PC after each instruction fetch so that the next instruction is fetched in sequence.
7. The fetched instruction is fetched into a register called instruction register (IR).

The basic instruction cycle is shown in the following figure:



Stage	Description of Stage	Simplified Description
1	The PC contains the address of the memory location that has the next instruction which has to be fetched	PC has address of next instruction
2	This address is then copied from the PC to the MAR via the address bus	PC copied to the MAR
3	The contents (instruction) at the memory location (address) contained in MAR are then copied into the MDR	Lookup MAR and get contents. Copy contents into the MDR
4	The contents (instruction) in the MDR is then copied and placed into the CIR	Copy MDR contents into the CIR
5	The value in the PC is then incremented by 1 so that it now points to the next instruction which has to be fetched	PC is then incremented by 1
6	The instruction is finally decoded and then executed by sending out signals (via control bus) to the various components of the computer	The instruction is decoded and then executed
7	Repeat	

After fetching an instruction, processor(s) execute the instruction by doing some processing on the data which may involve arithmetic and logic unit (ALU), specified by the instruction, then processor writes back the result (if any) to the memory.

The cycle of fetching, decoding, and executing an instruction is continually repeated by the CPU whilst the computer is turned on

The **type** of action the processor needs to take depends on which of the **4 categories** it falls into:

1. **Processor memory**: data may be transferred from processor to memory & vice versa
2. **Processor I/O**: data may be transferred to/from a peripheral device by transferring between the processor and an I/O module
3. **Data Processing**: The processor may perform some arithmetic or logic operation on data
4. **Control**: An instruction may specify that the sequence of execution be altered.

(E.g. the processor may fetch an instruction from location 149, which specifies that the next instruction be from location 182. The processor sets the program counter to 182. Thus, on the next fetch stage, the instruction will be fetched from location 182 rather than 150.)

An instruction's execution *may involve* a combination of these actions.

Program execution only HALTs if the processor is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the processor is encountered.

A major defining point in the history of computing was the realisation in 1944-1945 that data and instructions to manipulate the data were logically the same and could be stored in the same place.

The computer design built upon this principle, which became known as the *von Neumann architecture*, is still the basis for many computers today. Von Neumann architecture is serial in nature – so it matches with the ***fetch-execute (fetch-execute-decode) cycle***

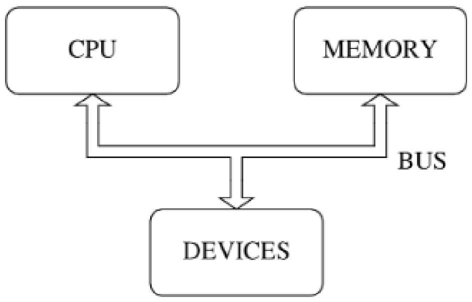
Although the name honours John Von Neumann (who also worked on the construction of the atomic bomb), the idea probably originated with Eckert and Mauchly, two other early pioneers who worked on the ENIAC at the Moore School at the University of Pennsylvania during the same period

Von Neumann Architecture

John von Neumann introduced a universal computer. It must comply with some principles and criteria:

- Computer consists of memory, arithmetical-logical unit (ALU), control unit, input and output devices
- All parts of a computer are connected together by Bus
- Computer structure is independent on the computed problem, a computer is programmed with content of memory
- Every computing step depends on the previous step
- Machine instruction and data are in the same memory, this is also known as ***stored program concept***.
- Memory is split to small cells with the same size. Their ordinal numbers are called address numbers
- Program consists of a sequence of instructions. Instructions are executed in order they are stored in memory.
- Instructions, characters, data and numbers are represented in binary form

The Von Neumann model combines the **Arithmetic and Logic Unit (ALU)** and the **Control Unit (CU)** into one functional unit, the **CPU.**



There is ***one*** bus used to transport data and instructions (which, by the way, are both stored in the same memory)

Data from ***memory*** and from ***input/output(I/O) devices*** are accessed ***in the same way***.

The Control Unit gets data and instruction in the same way from one memory. This simplifies its design and development.

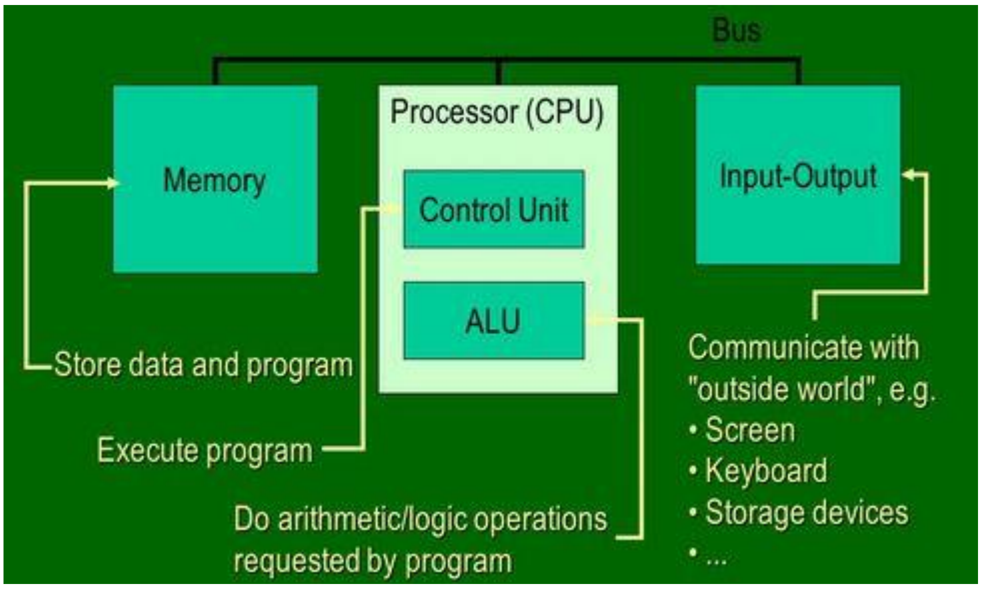
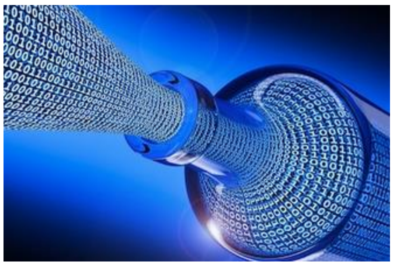


Figure 2 Von Neumann Architecture

CPU Bottleneck

With the Von Neumann architecture:



- It can process only one instruction at a time!!
- Instructions and data cannot be transferred at the same time = ***Von Neumann bottleneck***

- Instructions/data need to compete for control of the data bus and only one can be accessed at the same time,

Even though you have fast CPU and fast RAM there is a slowdown because only one or the other can be transferred at the same time.

Say a CPU is clocked at anywhere from 1.5 GHz to around 2.2GHz (for example the Apple A8 processor, which has two cores, runs at 1.4 GHz, in the iPhone 6… up to 50 times **faster** than the original iPhone).

But say the average RAM module is clocked at just 200MHz. So what that means is that the average bank of RAM is running with a clock speed that is a factor of 10 slower.

For the CPU this is an age to wait! When it requests something from RAM it has to wait and wait and wait while the data is fetched, time in which it could be doing something else, but can't as it needs to wait…

> The situation isn't actually that bad because of technologies like Double-Data-Rate (DDR) RAM which can send data twice per clock cycle. Likewise specifications like LPDDR3 (Low Power DDR3) allow for a data transfer rate eight times that of the internal clock. There are also techniques which can be built into the CPU which ensure that the data is requested as early as possible, before it is actually needed.

The problem is that modern processors uses 4 or 8 CPU cores, so there isn't just one CPU trying to access the memory, there are 8 of them and they all want that data, and they want it ASAP!

This performance limitation is known as the Von Neumann bottleneck.

Von Neumann was one of the key people in the invention of the modern day computer. The downside of the Von Neumann architecture is the performance bottleneck which appears when the data throughput is limited due to the relative speed differences between the CPU and the RAM.

RAM is slow, too slow, it just can't keep up with the processor. There are some methods to improve this situation and decrease the performance differential, one of which is the use of ***cache memory***.

Cache Memory

Cache is a small amount of memory that is built into the CPU chip that runs at the same speed as the CPU. This means that the CPU doesn't need to wait around for data from the cache memory, it is sent over to the CPU ***at the same speed*** that the CPU operates.

The cache memory is installed on a per CPU core basis, that means that each CPU core has ***its own cache*** memory and there won't be any contention about who gets access to it.

Since it only has a few Kilobytes of cache memory (because of the price) available there will be times when the cache has the right memory contents, known as a ***hit***, and times when it doesn't, known as a ***miss***. The more cache hits the better.

Splitting Cache

To help improve the number of hits versus misses there are a number of techniques which are used. One is to divide the cache in two, one for instructions and one for data.

Another technique to improve overall cache hits is to use a hierarchy of caches, these are traditionally known as L1 (level 1) and L2 (level 2) caches.

The Harvard Architecture

Having discussed the Von Neumann bottleneck, we come to another alternative architecture that helps to alleviate the bottleneck called the ***Harvard architecture***.

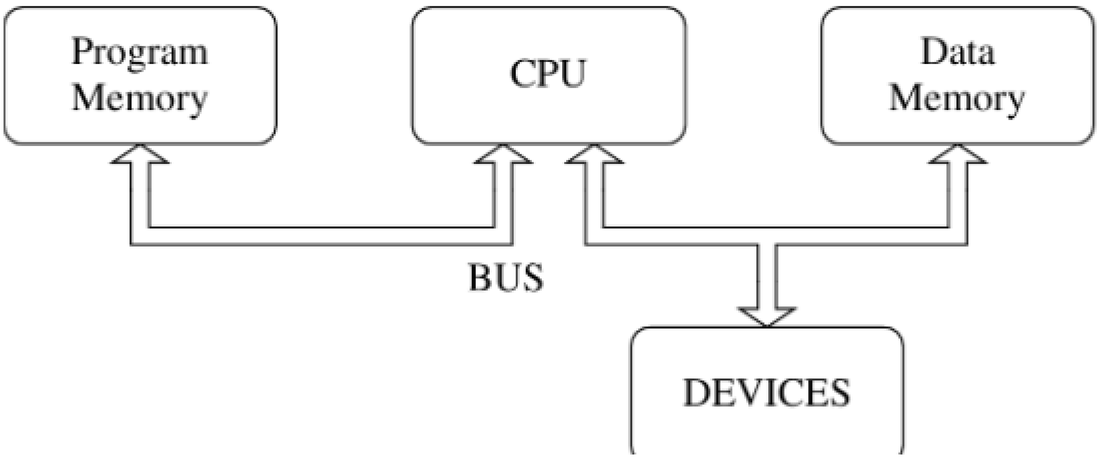
The MARK II computer was finished at Harvard University in 1947 (research online and read about the first computer bug!!). It wasn't as modern as the computer from von Neumann team. But it introduced a slightly different architecture. Memory for data was separated from the memory for instruction. This concept is known as the ***Harvard architecture***.

Harvard architecture is a different structure than Von Neumann architecture in which it has a ***physically separate storage and signal pathways (bus) for instructions and data***

Von Neumann only has one bus for both instructions and data and so the Harvard architecture can do both at the same time whereas the Von Neumann has to schedule them and can't do both of them at the same time.

This allows parallel access to instructions and data

It is also possible to have two separate memory systems in Harvard architecture so long as the data and instructions can be inputted at the same time then it can come from the memory or the cache, which can also use different cell sizes.



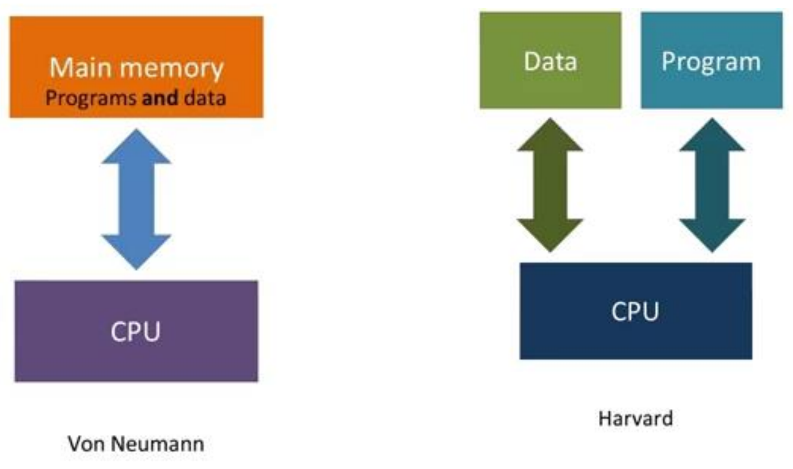
There is a (physical) separate memory for instructions and data, which means that production of a computer with two buses is more expensive and needs more time. The Control Unit therefore for two buses is more complicated and more expensive.

So, for Harvard Architecture:

➔ Each memory has its own bus

Harvard architecture will be faster than Von Neumann as it’s avoiding the Von Neumann bottleneck

➔ CPU using Harvard architecture can access instructions and read/write data at the same time



Von Neumann architecture is similar to the Harvard architecture except it uses a single bus to perform both instruction fetches and data transfers, so the operations must be scheduled.

The Harvard architecture, on the other hand, uses two separate memory addresses for data and instructions, which makes it possible to feed data into both the busses at the same time.

However, the complex architecture only adds to the development cost of the control unit against the lower development cost of the less complex Von Neumann architecture which employs a single unified cache.

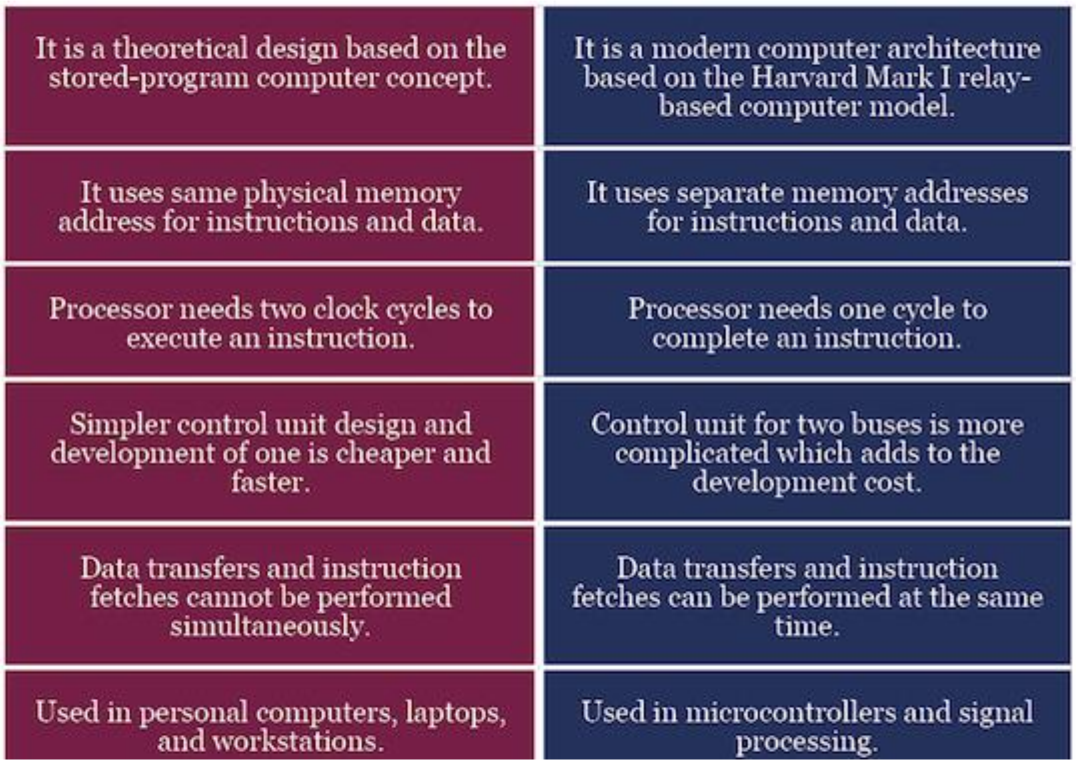
Application of Architectures

The Von Neumann architecture is better for computers, laptops, workstations and high performance computers.

The Harvard architecture is used primarily for small embedded computers, and in reality, most modern computers instead implement a *modified* Harvard architecture. The modified version includes modifications which are various ways to loosen the strict separation between code and data, while still supporting the higher performance concurrent data and instruction access of the Harvard architecture.

A modified Harvard architecture machine is very much like a Harvard architecture machine, but it relaxes the strict separation between instruction and data while still letting the CPU concurrently access two (or more) memory buses.

In Summary



It is a theoretical design based on the stored-program computer concept.	It is a modern computer architecture based on the Harvard Mark I relay-based computer model.
It uses same physical memory address for instructions and data.	It uses separate memory addresses for instructions and data.
Processor needs two clock cycles to execute an instruction.	Processor needs one cycle to complete an instruction.
Simpler control unit design and development of one is cheaper and faster.	Control unit for two buses is more complicated which adds to the development cost.
Data transfers and instruction fetches cannot be performed simultaneously.	Data transfers and instruction fetches can be performed at the same time.
Used in personal computers, laptops, and workstations.	Used in microcontrollers and signal processing.

Figure 3 Von Neumann Architecture Versus Harvard Architecture

The Future…???

With the rapid progress of semiconductor technology, billions of transistors can be integrated on a single silicon chip.

However, increasing power consumption, complicated system function and huge investment will slow down the development of process technology and impact the integrated circuit products.