

CIS-77 Home <http://www.c-jump.com/CIS77/CIS77syllabus.htm>

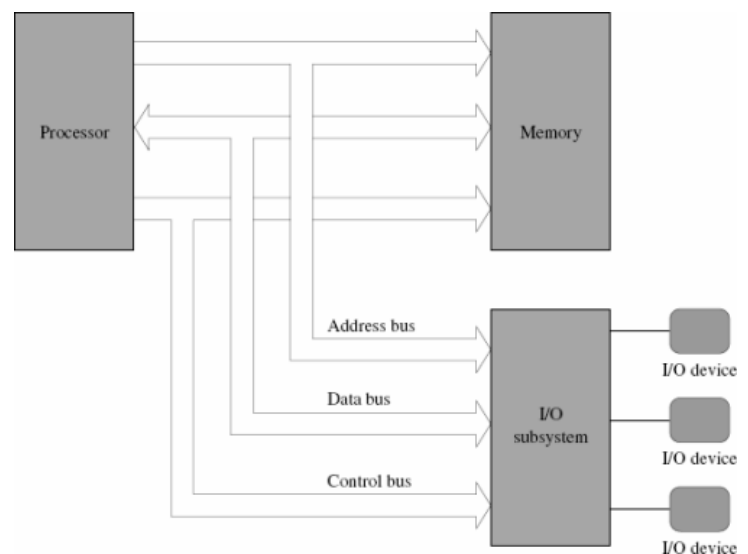
## The von Neumann Computer Model

1. [The von Neumann Computer Model](#)
2. [Components of the Von Neumann Model](#)
3. [Communication Between Memory and Processing Unit](#)
4. [CPU data-path](#)
5. [Memory Operations](#)
6. [Understanding the MAR and the MDR](#)
7. [Understanding the MAR and the MDR, Cont.](#)
8. [ALU, the Processing Unit](#)
9. [ALU and the Word Length](#)
10. [Control Unit](#)
11. [Control Unit, Cont.](#)
12. [Input/Output](#)
13. [Input/Output Ports](#)
14. [Input/Output Address Space](#)
15. [Console Input/Output in Protected Memory Mode](#)
16. [Instruction Processing](#)
17. [Instruction Components](#)
18. [Why Learn Intel x86 ISA ?](#)
19. [Design of the x86 CPU Instruction Set](#)
20. [CPU Instruction Set](#)
21. [History of IBM PC](#)
22. [Early x86 Processor Family](#)
23. [8086 and 8088 CPU](#)
24. [80186 CPU](#)
25. [80286 CPU](#)
26. [80386 CPU](#)
27. [80386 CPU, Cont.](#)
28. [80486 CPU](#)
29. [Pentium \(Intel 80586\)](#)
30. [Pentium Pro](#)
31. [Pentium II](#)
32. [Itanium processor](#)

### 1. The von Neumann Computer Model

- Von Neumann computer systems contain three main building blocks:
  - the central processing unit (CPU),
  - memory,
  - and input/output devices (I/O).
- These three components are connected together using the *system bus*.
- The most prominent items within the CPU are the registers: they can be manipulated directly by a computer program.

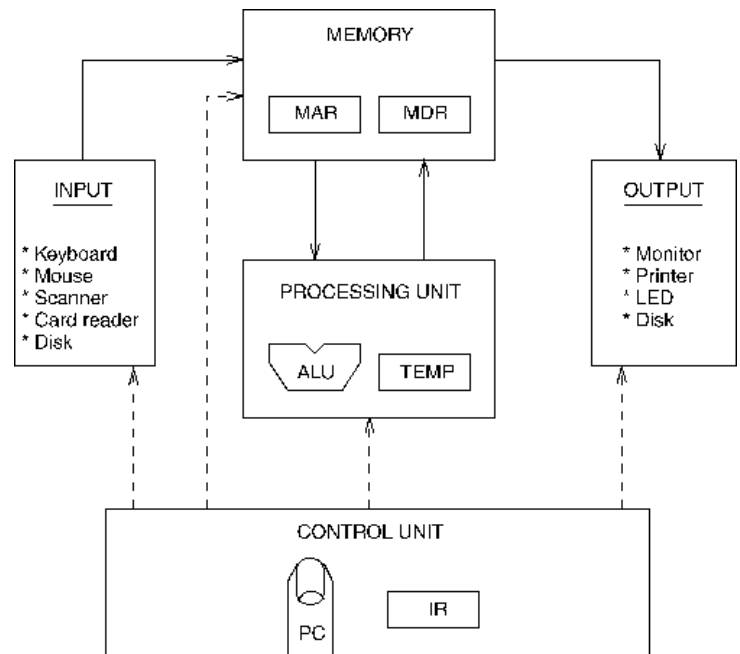
The following block diagram shows major relationship between CPU components:



## 2. Components of the Von Neumann Model

1. **Memory:** Storage of information (data/program)
2. **Processing Unit:** Computation/Processing of Information
3. **Input:** Means of getting information into the computer. e.g. keyboard, mouse
4. **Output:** Means of getting information out of the computer. e.g. printer, monitor
5. **Control Unit:** Makes sure that all the other parts perform their tasks correctly and at the correct time.

The von Neumann Machine:



## 3. Communication Between Memory and Processing Unit

Communication between memory and processing unit consists of two *registers*:

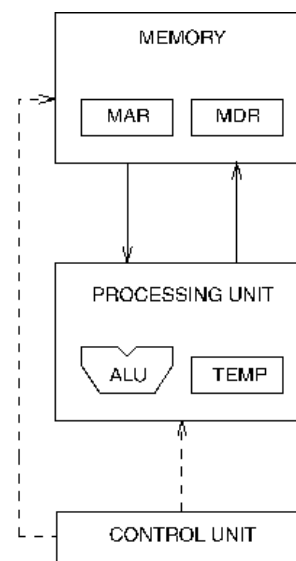
- Memory Address Register (MAR).
- Memory Data Register (MDR).

To read,

1. The address of the location is put in MAR.
2. The memory is *enabled* for a read.
3. The value is put in MDR by the memory.

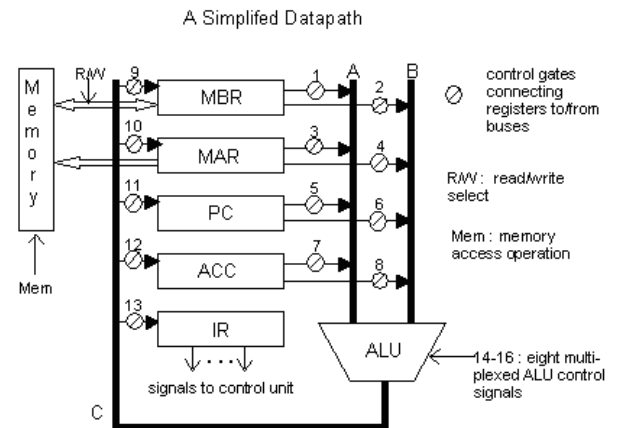
To write,

1. The address of the location is put in MAR.
2. The data is put in MDR.
3. The **Write Enable** signal is *asserted*.
4. The value in MDR is written to the location specified.

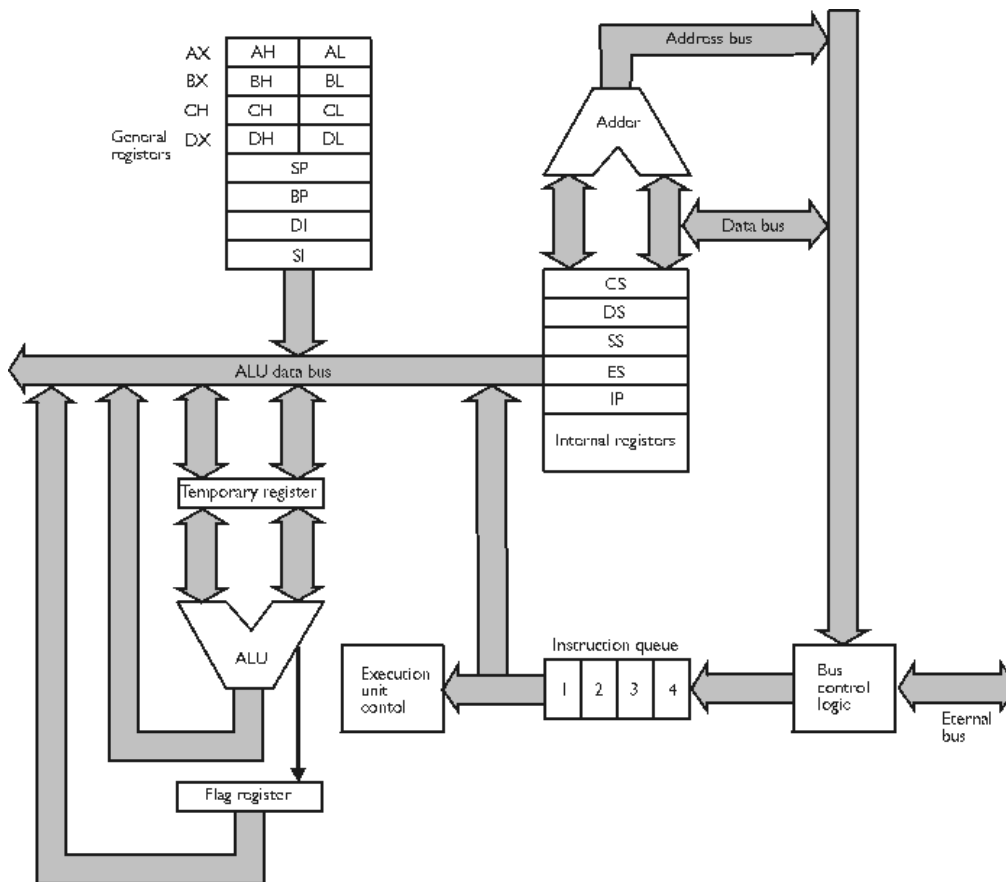


## 4. CPU data-path

- Hardware units like **ALU's**, **registers**, **memory**, etc., are linked together into a *data-path*.
- The flow of bits around the data-path is controlled by the "gates" which allow the bits to flow (on) or not flow (off) through the data-path.
- The binary instructions (1 = on; 0 = off) that control the flow are called *micro-instructions*.



Simplified [x86 data path](#) :

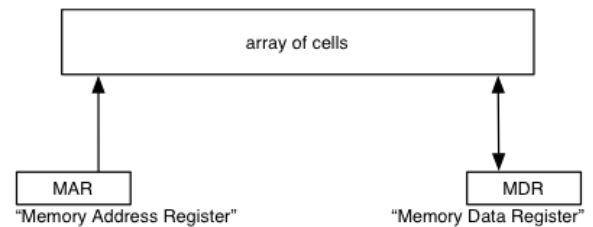


## 5. Memory Operations

- There are two key operations on memory:

Memory Operations:

1. **fetch( address )** returns value without changing the value stored at that address.
  2. **store( address, value )** writes new value into the cell at the given address.
- This type of memory is *random-access*, meaning that CPU can access any value of the array at any time (vs. sequential access, like on a tape).
  - Such memories are called **RAM** (random-access memory.)
  - Some memory is non-volatile, or read-only (**ROM** or read-only memory.)



**fetch (addr):**

1. Put *addr* into MAR
2. Tell memory unit to "load"
3. Memory copies data into MDR

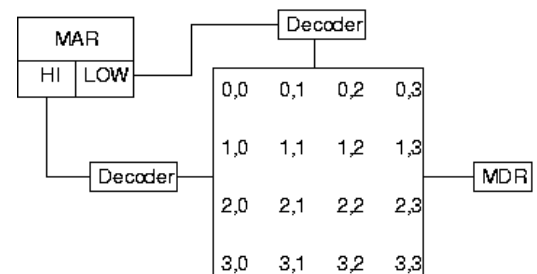
**store (addr, new-value):**

1. Put *addr* into MAR
2. Put *new-value* into MDR
3. Tell memory unit to "store"
4. Memory stores data from MDR into memory cell.

## 6. Understanding the MAR and the MDR

- **MAR** stands for *memory address register*:
  - **MAR** is connected to the [address bus](#).
  - **MAR** is "the only way" for the CPU to communicate with [address bus](#).
  - [Tri-state buffer](#) between **MAR** and the address bus prevents **MAR** from continuously dumping its output to the address bus.
  - **MAR** can hold either an *instruction address* or a *data address*.

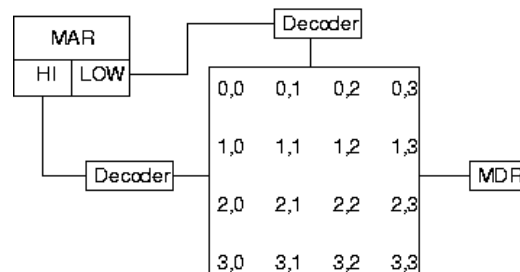
Memory Circuitry:



## 7. Understanding the MAR and the MDR, Cont.

- **MDR** Stands for *memory data register*.
  - **MDR** is connected to the [data bus](#).
  - Data can go in both directions: to and from memory, therefore, **MDR** can load its data from
    - the [data bus](#) (for reading data)
    - one of the CPU registers (for storing data.)
  - A **2-1 MUX** circuit selects input from one of the two.

Memory Circuitry:

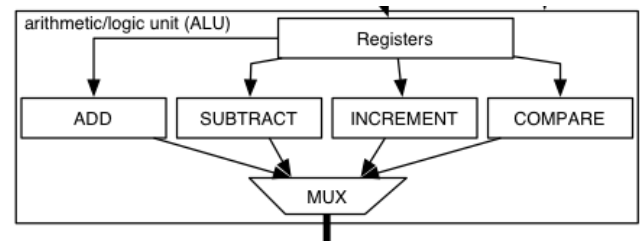


More info: [von Neumann Architecture](#) article by Prof. Fabio Pellacini, Dartmouth College, NH.

## 8. ALU, the Processing Unit

- Processing unit is hardware that implements Arithmetic and Logical Operations.
- ALU stands for *Arithmetic and Logic Unit*, capable of performing `ADD`, `SUBTRACT`, `AND`, `OR`, and `NOT` operations.

The ALU:



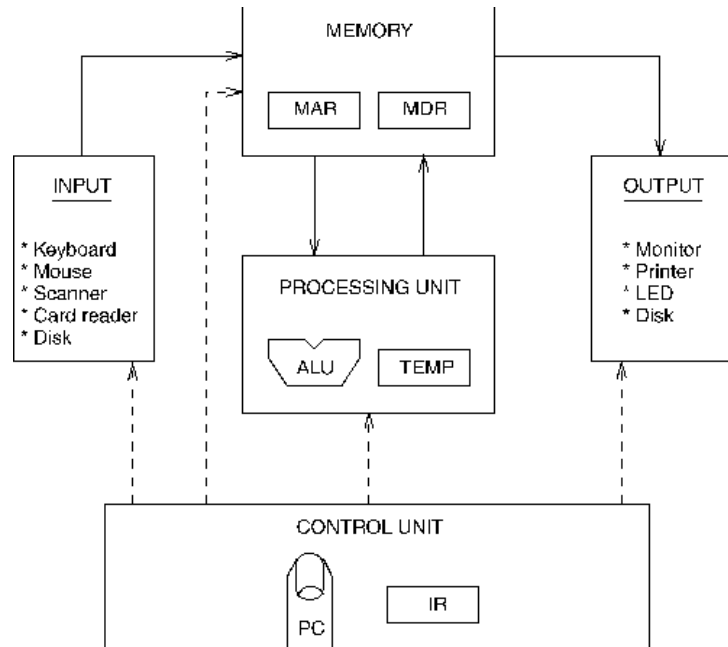
- The size of input quantities of ALU is often referred to as *word length* of the computer.
- Many processors today have word length of 32 and 64 bit.
- Processing unit also includes a set of **registers** for *temporary storage* of data and memory addressing.

## 9. ALU and the Word Length

- The size of quantities processed by ALU is the *word length* of the computer.
- The word length does not affect what a computer can compute.
- A computer with a smaller word length can do *the same* computation as one with a larger word length...  
...but it will take more time.
- For example, to add two 64 bit numbers,
  - word length = 16 takes 4 adds.
  - word length = 32 takes 2 adds.
  - word length = 64 takes 1 add.

## 10. Control Unit

- Manages the Precessing Unit.
- Implemented as FSM.
- FSM directs all activity.
- Clock-based step-by-step precessing, cycle-by-cycle.
- FSM is controlled by the
  1. Clock signal
  2. Instruction Register
  3. Reset signal



## 11. Control Unit, Cont.

Control unit includes

- Instruction Register **IR**
- Instruction Pointier **IP** (aka Program Counter **PC**)

FSM *outputs* of the control unit,

- - - - ->

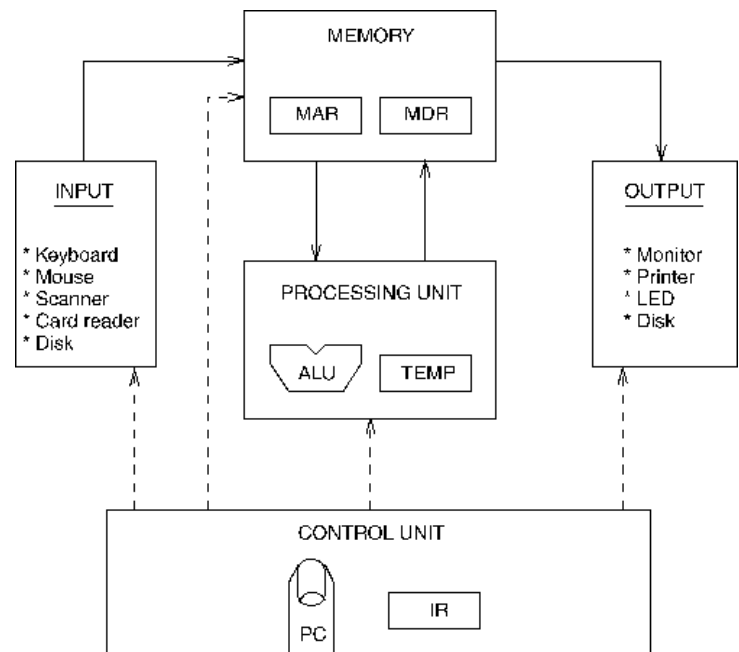
shown by dash lines, have two purposes:

1. Control processing that takes place inside the ALU.
2. Authorize read/write gate control of the CPU data-path.

Control unit demo: [cjumpcpx.exe](#) ->

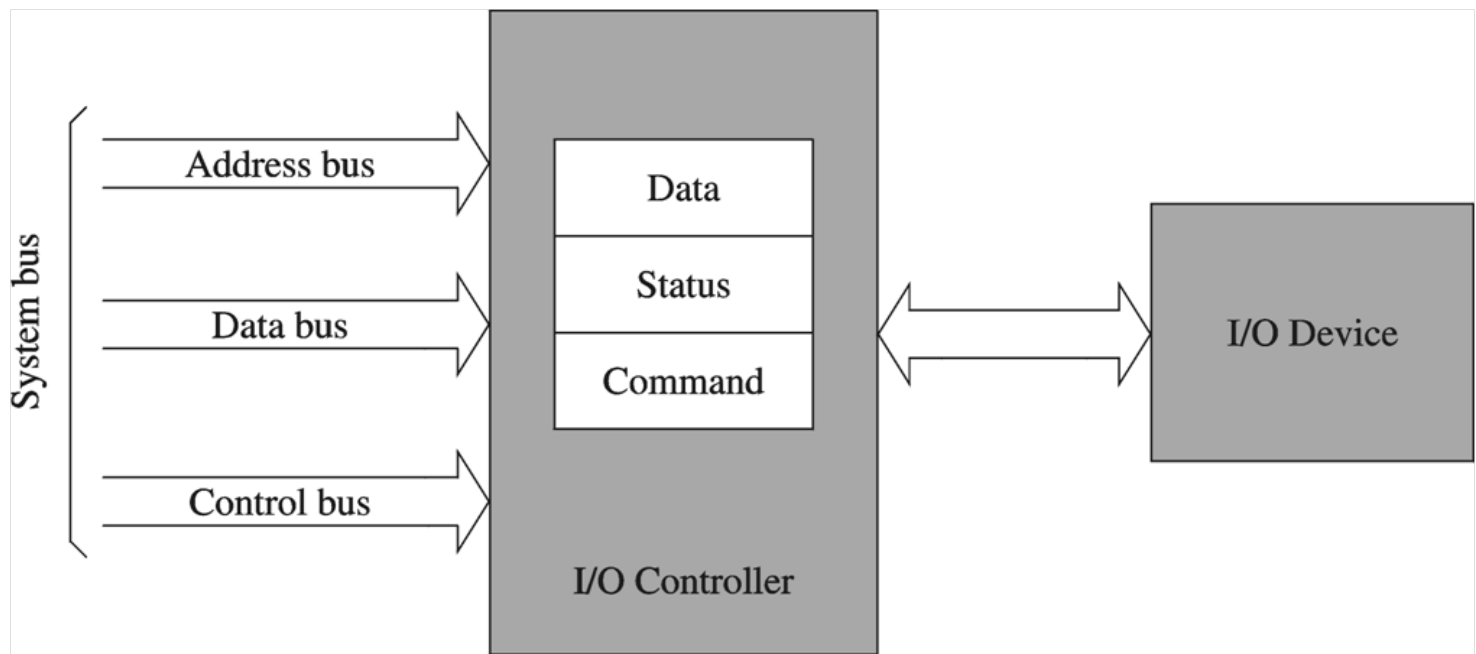
**CPU, memory, Assembly** ->

**Protected memory.**



## 12. Input/Output

- I/O controller provides the necessary interface to I/O devices.
- Takes care of low-level, device-dependent details.
- Provides necessary electrical signal interface.



### 13. Input/Output Ports

- Processor and I/O interface points for exchanging data are called **I/O ports**.
- Two ways of *mapping* I/O ports:
  1. **Memory-mapped I/O:**
    - I/O ports are mapped to the memory address space.
    - Reading/writing I/O is similar to reading/writing memory.
    - Can use memory read/write instructions.
    - Motorola 68000 uses memory-mapped I/O.
  2. **Isolated I/O:**
    - Separate *I/O address space*.
    - Intel 80x86 processors support isolated I/O.
    - Requires special I/O instructions, like **IN** and **OUT** on x86.

### 14. Input/Output Address Space

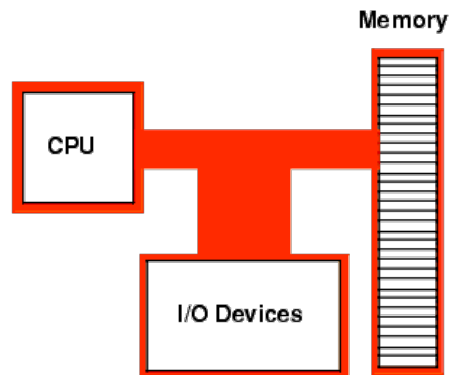
- Pentium x86 provides 64 KB of **I/O address space**.
- Can be used for 8-, 16-, and 32-bit I/O ports.
- Combination cannot exceed the total I/O address space, can have either
  - 64 K 8-bit ports
  - 32 K 16-bit ports
  - 16 K 32-bit ports

- A combination of these for a total of 64 KB.
- **I/O instructions** do not go through memory *segmentation* or *paging*.
- **I/O address** refers to the physical I/O address.

## 15. Console Input/Output in Protected Memory Mode

- Using console in Protected Memory Mode:
  - Keyboard input
  - Console display output
  - Ascii character encoding

Von Neumann Computer System Block Diagram:



## 16. Instruction Processing

Central idea of von Neumann model is that both program and data stored in computer memory:

- Program is a sequence of instructions
- Instruction is a binary encoding of operations and operands:
- For example, an arithmetic expression

$$-a + b * c$$

could be computed by a program with three machine instructions

```
neg ax      ; negate (multiply by -1)
imul bx, cx ; multiply and store result in bx
add ax, bx  ; add and store result in ax
```

where **NEG**, **IMUL**, **ADD** are arithmetic instructions, **AX**, **BX**, and **CX** are operands.

- Instructions are most basic units of processing.
- Instructions are executed under control of the control unit.

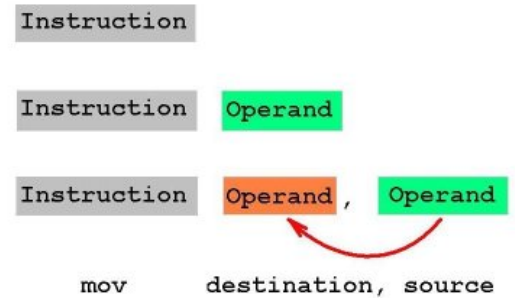
## 17. Instruction Components

- Instruction in memory has two parts: opcode and operands.
- The opcode identifies the operation that instruction does
- The operands are subjects of the operation, such as data values, registers, or memory addresses.

Instruction formats:



- Due to variety of opcodes and operands, instructions may occupy different sizes of bytes in memory.
- Instructions with no operands can have *implied operands*, those that are not explicitly shown.



## 18. Why Learn Intel x86 ISA ?

- The x86 Intel CPU family is generally classified as a *Von Neumann Architecture Machine*.
- Most pervasive ISA in the marketplace.
- Opcodes have zero or more operands.
- Instructions and operands have strong support for *data types*.
- All x86 Assemblers are free, including Microsoft Macro Assembler, [MASM](#).
- Fullscreen 32-bit debuggers **WinDbg**, **OllyDbg**, and **MS Visual Studio**.

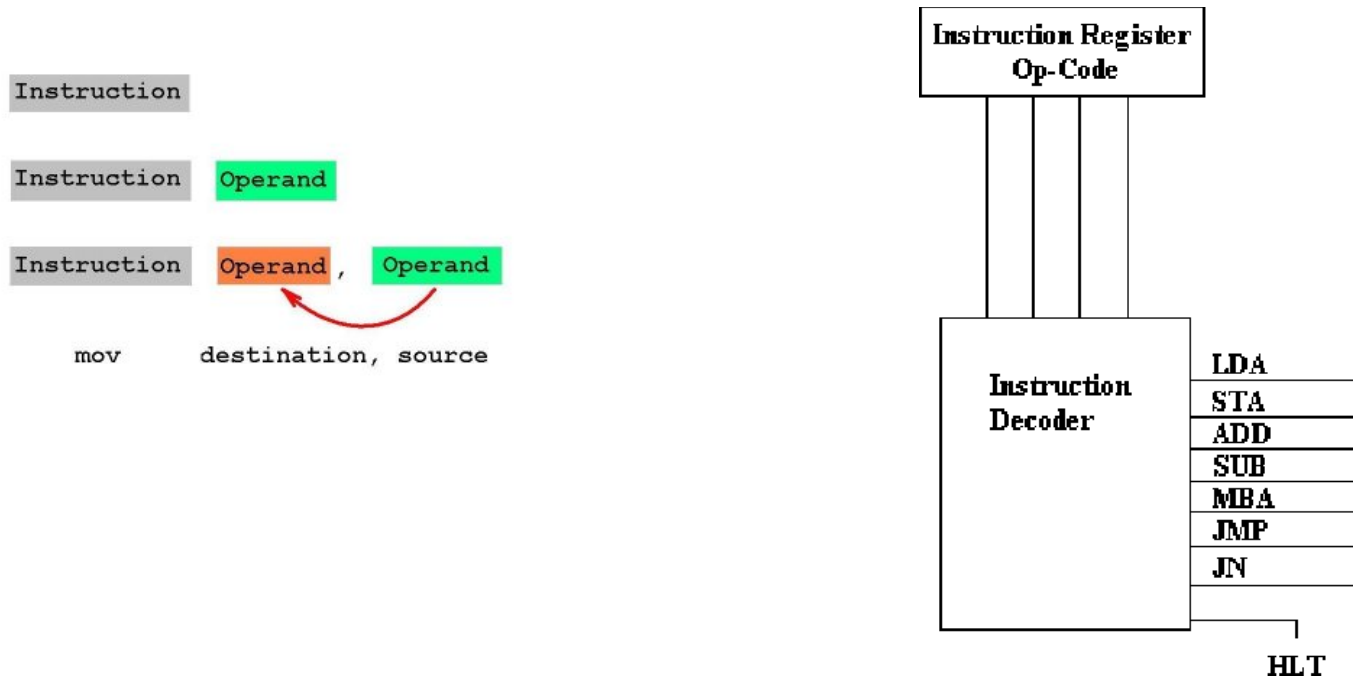
## 19. Design of the x86 CPU Instruction Set

- *Instruction set architecture* (ISA) is an important design issue for CPU designers:
  - caches, pipelining, superscalar* implementation can be added to the design later, but it is very difficult to change the CPU instructions once in production and people writing software...
  - ...[LOOP](#) instruction on the x86 CPU sees very little use in modern high-performance programs.
- Challenges:
  - **Expandability** is a big concern (predicting future needs.)
  - **Legacy support** is almost the opposite of expandability.
- Each instruction requires some number of transistors on the CPU's silicon die.

## 20. CPU Instruction Set

- A typical Von Neumann architecture encodes CPU instructions as numeric values in memory. Instruction decoder:
- Programming and engineering of the *instruction set* is a major task in the CPU design.
- Each instruction needs a *unique numeric opcode*.

Instruction formats:



## 21. History of IBM PC

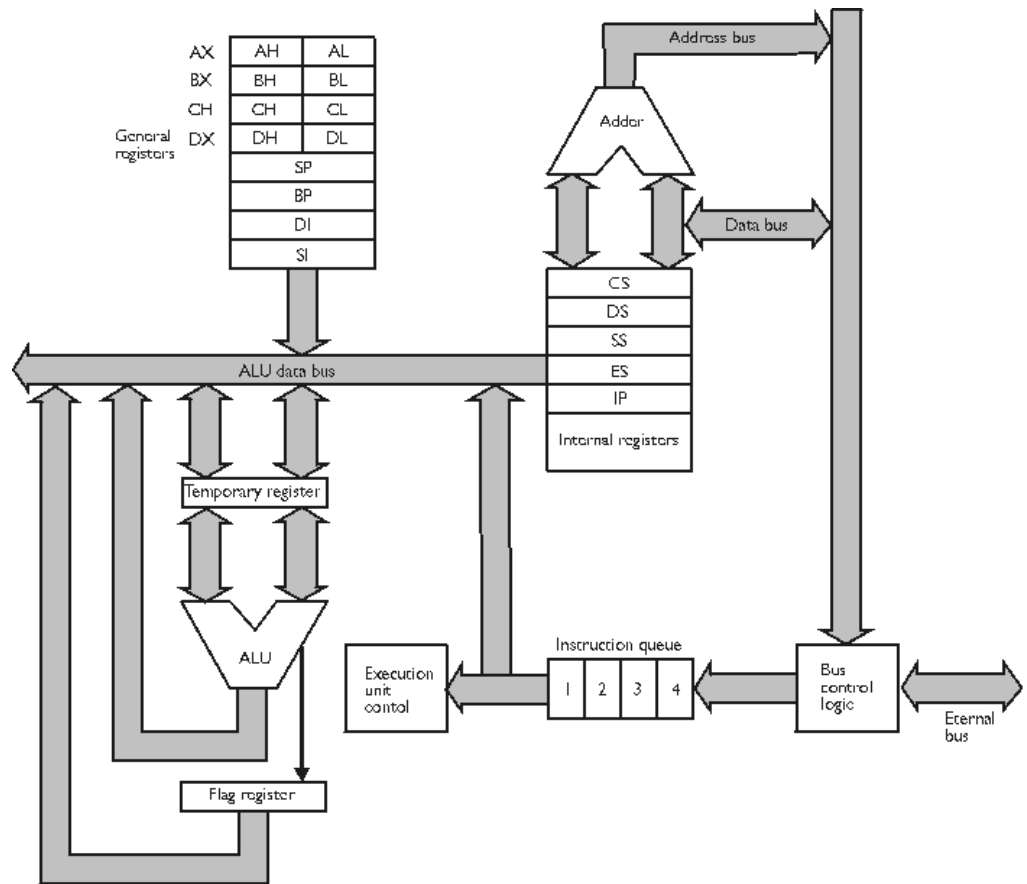
- Design a simple instruction set to begin with and leave room for later **expansion** -  
main reason the x86 is so popular and long-lived.
- Intel started with a relatively simple CPU and figured out how to extend the instruction set to accommodate new features.
- IBM decided that open documentation is a *good thing*.
- IBM published the [IBM PC Technical Reference Manual](#), with everything the engineers knew about the machine, which was another reason for the success of the IBM PC platform.
- See also: [25 Years of the IBM PC](#) and CNN article [Meet the inventor of Ctrl-Alt-Delete](#)

## 22. Early x86 Processor Family

- Intel introduced microprocessors in 1969: 4-bit microprocessor 4004.
- 8-bit microprocessors are 8080 and 8085 models.
- 16-bit processors:
  - 8086 introduced in 1979.
  - 20-bit address bus, 16-bit data bus.  
(8088 is a less expensive version.)
  - Uses 8-bit data bus.
  - Can address up to 4 [segments](#) of 64 KB.
  - Referred to as the [real mode addressing](#).
  - Still fully compatible with modern x86 chips.

## 23. 8086 and 8088 CPU

- Year 1979
- The 8088, 8086, and 80286 are 16-bit CPUs.
- Internal registers are 16 bits in size.
- The 8086 is faster than the 8088 because of its 16-bit data bus; the 8088 has only an 8-bit data bus.
- The 16-bit data bus allows you to use EVEN and ALIGN on an 8086 processor to word-align data and thus improve data-handling efficiency.
- Memory addresses on the 8086 and 8088 refer to actual physical addresses.
- The 8086 and 8088 have 20 address pins, and 1 megabyte of addressable memory (which is the real mode segmented memory explained later) requires addresses of 20 bits in size.
- See also: [Brief History of Microprocessors](#)



## 24. 80186 CPU

- A [faster](#) version of 8086.
- [16-bit data bus](#) and [20-bit address bus](#).
- Improved instruction set.

## 25. 80286 CPU

- Year 1982.
- Introduced [protected mode addressing](#).
- Enhanced with memory protection capabilities with *some instructions to control protected mode*.
- Runs [faster](#).
- 24-bit address bus.
- 16 MB address space.

- [Segmentation](#) in protected mode is different from the [real mode](#).
- Allows the operating system to run [multiple processes](#) at the same time.
- The 80286 was the minimum for running Windows 3.1 and higher.
- Backwards compatible.

## 26. 80386 CPU

---

- Year 1985.
- The 80386 and many of its successors are [32-bit](#) CPUs.
- Internal registers are 32 bits in size.
- Unlike its predecessors, the 80386 processor can handle both 16-bit and 32-bit data.
- Supports entire instruction set of 80286.
- Adds several [new instructions](#) as well.
- Software written for the 80286 runs unchanged on the 80386, but the chip is [faster](#).
- The 80386 implements many new hardware-level features, including
  - [paged memory](#).
  - segmentation can be turned off ([flat model](#)).
  - support for multiple virtual 8086 processes.
  - addressing of up to [4 gigabytes](#) of memory.
  - specialized debugging registers.

## 27. 80386 CPU, Cont.

---

- 32bit operating systems such as Windows NT required 80386 or higher processor.
- More embedded systems nowadays use the 80386 CPU.
- Examples include:
  - high speed data communications devices.
  - graphics equipment.
  - tra-high-speed data acquisition gear.
  - Cockpit displays of some modern jetliners use 80386 as controllers.
- See also: [Intel 386 Manuals](#)

## 28. 80486 CPU

---

- Year 1989.
- The 80486 processor is an enhanced version of the 80386 with
  - instruction *pipelining* allows [parallel execution](#) capability to instruction decode and execution units.
  - executes many instructions 2-3 times faster.
  - achieves scalar execution of 1 instruction per clock cycle.

- The chip incorporates
  - [math coprocessor](#) for performing floating-point arithmetic
  - 8 Kilobyte memory cache.

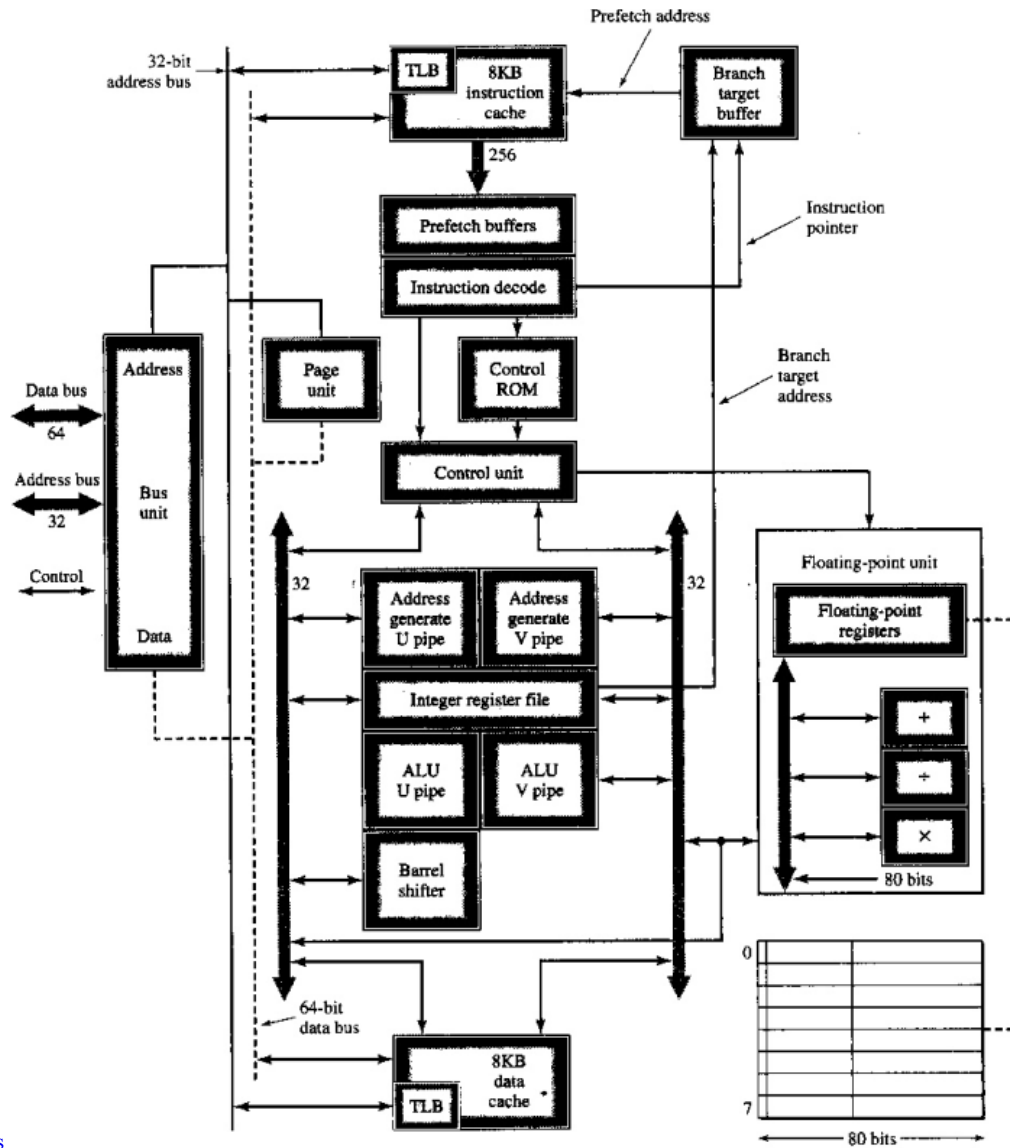
(The math coprocessor was disabled on a variation of the chip called the 80486SX.)

- New instructions.
- Fully compatible with 80386 software.
- Later versions introduced [energy savings](#) for laptops.
- See also: [Intel 486 Manuals](#)

## 29. Pentium (Intel 80586)

---

- Pentium was introduced in 1993, followed by
  - Pentium Pro in 1995,
  - Pentium II in 1997,
  - Pentium III in 1999,
  - Pentium IV 2001.
- Similar to 80486 but with [64-bit data bus](#).
- Wider internal datapaths: 128- and 256-bit wide.
- Added second execution pipeline.
- Superscalar performance allows two instructions per clock cycle.
- Doubled on-chip level-1 cache:
  - one 8 KB for data
  - another 8 KB for instructions.
- Added branch prediction.



- See also: [Pentium Manuals](#)

### 30. Pentium Pro

- Introduced in 1995.
- Three-way superscalar performance, 3 instructions per clock cycle.
- 36-bit address bus.
- 64 GB address space.
- Introduced dynamic execution:
  - Out-of-order execution.
  - Speculative execution.
- In addition to the level-1 cache, has 256 KB level-2 cache.

### 31. Pentium II

- Introduced in 1997.
- Added multimedia (MMX) instructions.
- Doubled on-chip level-1 cache:
  - 16 KB for data
  - 16 KB for instructions.
- Introduced comprehensive power management features:
  - Sleep
  - Deep sleep.
- In addition to the level-1 cache has 256 KB level-2 cache.

### 32. Itanium processor

---

- RISC design  
(All previous x86 designs were CISC.)
- 64-bit processor
- Uses 64-bit address bus
- 128-bit data bus
- Introduced several advanced features:
  - Speculative execution
  - Predication to eliminate branches
  - Branch prediction