**MICROPROCESSOR**

**Generation**

**Processor Address Bus Data Bus Clock Speed  
  
4004 10 4 108KHz  
8008 14 8 200KHz  
8080 16 8 2MHz  
8085 16 8 5MHz  
8086 20 16 5MHz  
8088 20 8 5MHz  
80286 24 16 8MHz  
80386 32 32 16MHz  
80486 32 32 25MHz  
Pentium 32 32/64 60MHz  
Pentium Pro 36 32/64 150MHz  
Pentium II 36 64 233MHz  
Pentium III 36 64 650MHz  
Pentium 4 36 64 1.4GHz**

**Microcontroller** - A **microcontroller** (sometimes abbreviated **µC**, **uC** or **MCU**) is a small computer on a single [integrated circuit](http://en.wikipedia.org/wiki/Integrated_circuit) containing a [processor core](http://en.wikipedia.org/wiki/Central_processing_unit), memory, and programmable [input/output](http://en.wikipedia.org/wiki/Input/output) peripherals. Program memory in the form of [NOR flash](http://en.wikipedia.org/wiki/NOR_flash) or [OTP ROM](http://en.wikipedia.org/wiki/Programmable_read-only_memory) is also often included on chip, as well as a typically small amount of [RAM](http://en.wikipedia.org/wiki/Random-access_memory). Microcontrollers are designed for embedded applications, in contrast to the [microprocessors](http://en.wikipedia.org/wiki/Microprocessor) used in [personal computers](http://en.wikipedia.org/wiki/Personal_computer) or other general purpose applications. Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other [embedded systems](http://en.wikipedia.org/wiki/Embedded_system). By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes.

**Word** – The word or word length is defined as the number of bits the microprocessor recognizes and processes at a time.

**Nibble -** In [computing](http://en.wikipedia.org/wiki/Computing), a **nibble** (often **nybble** or even **nyble** to match the vowels of [**byte**](http://en.wikipedia.org/wiki/Byte)) is a four-[bit](http://en.wikipedia.org/wiki/Bit) aggregation,or half an [octet](http://en.wikipedia.org/wiki/Octet_%28computing%29). As a nibble contains 4 bits, there are sixteen (24) possible values, so a nibble corresponds to a single [hexadecimal](http://en.wikipedia.org/wiki/Hexadecimal) digit (thus, it is often referred to as a "hex digit" or "hexit").

A full [byte](http://en.wikipedia.org/wiki/Byte) (octet) is represented by two hexadecimal digits; therefore, it is common to display a byte of information as two nibbles. The nibble is often called a "semioctet" or a "quartet" in a [networking](http://en.wikipedia.org/wiki/Computer_network) or [telecommunication](http://en.wikipedia.org/wiki/Telecommunication) context.

[**Instruction set**](http://www.webopedia.com/TERM/I/instruction.html): The set of instructions that the microprocessor can execute.

[**Bandwidth**](http://www.webopedia.com/TERM/B/bandwidth.html) **:** The number of [bits](http://www.webopedia.com/TERM/B/bit.html) processed in a single instruction.

[**Clock speed**](http://www.webopedia.com/TERM/C/clock_speed.html) **:** Given in megahertz ([MHz](http://www.webopedia.com/TERM/M/MHz.html)), the clock speed determines how many instructions per second the [processor](http://www.webopedia.com/TERM/P/processor.html) can [execute](http://www.webopedia.com/TERM/E/execute.html).

Note: the higher the value, the more powerful the CPU. For example, a [32-bit](http://www.webopedia.com/TERM/3/32_bit.html) microprocessor that [runs](http://www.webopedia.com/TERM/R/run.html) at 50MHz is more powerful than a 16-bit microprocessor that runs at 25MHz.

In addition to bandwidth and clock speed, microprocessors are classified as being either [RISC](http://www.webopedia.com/TERM/R/RISC.html) (reduced instruction set [computer](http://www.webopedia.com/TERM/C/computer.html)) or [CISC](http://www.webopedia.com/TERM/C/CISC.html) (complex instruction set computer).

**Assembler:** A computer will not understand any program written in a language, other than its machine language. The programs written in other languages must be translated into the machine language. Such translation is performed with the help of software. A program which translates an assembly language program into a machine language program is called an assembler. If an assembler which runs on a computer and produces the machine codes for the same computer then it is called **Self assembler or Resident assembler.** If an assembler that runs on a computer and produces the machine codes for other computer then it is called **Cross Assembler.**

Assemblers are further divided into two types: **One Pass Assembler and Two Pass Assembler.** One pass assembler is the assembler which assigns the memory addresses to the variables and translates the source code into machine code in the first pass simultaneously. A Two Pass Assembler is the assembler which reads the source code twice. In the first pass, it reads all the variables and assigns them memory addresses. In the second pass, it reads the source code and translates the code into object code.

**Compiler:** It is a program which translates a high level language program into a machine language program. A compiler is more intelligent than an assembler. It checks all kinds of limits, ranges, errors etc. But its program run time is more and occupies a larger part of the memory. It has slow speed. Because a compiler goes through the entire program and then translates the entire program into machine codes.

If a compiler runs on a computer and produces the machine codes for the same computer then it is known as a **Self compiler or Resident compiler.** On the other hand, if a compiler runs on a computer and produces the machine codes for other computer then it is known as a **Cross compiler.**

**Interpreter:** An interpreter is a program which translates statements of a program into machine code. It translates only one statement of the program at a time. It reads only one statement of program, translates it and executes it. Then it reads the next statement of the program again translates it and executes it. In this way it proceeds further till all the statements are translated and executed. On the other hand, a compiler goes through the entire program and then translates the entire program into machine codes**. A compiler is 5 to 25 times faster than an interpreter.**

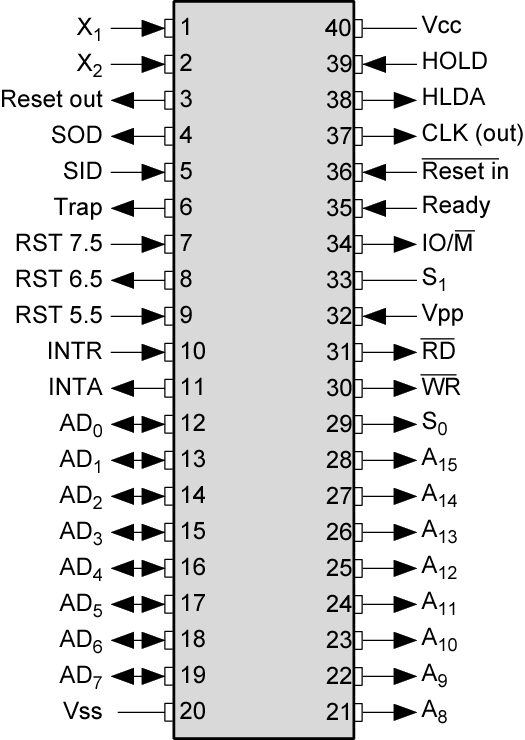
By the compiler, the machine codes are saved permanently for future reference. On the other hand, the machine codes produced by interpreter are not saved. An interpreter is a small program as compared to compiler. It occupies less memory space, so it can be used in a smaller system which has limited memory space.

**Linker:** In high level languages, some built in header files or libraries are stored. These libraries are predefined and these contain basic functions which are essential for executing the program. These functions are linked to the libraries by a program called Linker. If linker does not find a library of a function then it informs to compiler and then compiler generates an error. The compiler automatically invokes the linker as the last step in compiling a program.

Not built in libraries, it also links the user defined functions to the user defined libraries. Usually a longer program is divided into smaller subprograms called modules. And these modules must be combined to execute the program. The process of combining the modules is done by the linker.

**Loader:** Loader is a program that loads machine codes of a program into the system memory. In Computing, a **loader** is the part of an Operating System that is responsible for loading programs. It is one of the essential stages in the process of starting a program. Because it places programs into memory and prepares them for execution.

Loading a program involves reading the contents of executable file into memory.  Once loading is complete, the operating system starts the program by passing control to the loaded program code. All operating systems that support program loading have loaders. In many operating systems the loader is permanently resident in memory.



**The typical processor system consists of:**

* + *CPU (central processing unit)*
    - *ALU (arithmetic-logic unit)*
    - *Control Logic*
    - *Registers, etc…*
  + *Memory*
  + *Input / Output interfaces*

**Interconnections between these units:**

* + Address Bus
  + Data Bus
  + Control Bus

**The 8085: CPU Internal Structure**

***Registers***

* + *Six general purpose 8-bit registers: B, C, D, E, H, L*
  + *They can also be combined as register pairs to perform 16-bit operations: BC, DE, HL*
  + *Registers are programmable (data load, move, etc.)*

***Accumulator***

* + *Single 8-bit register that is part of the ALU.*
  + *Used for arithmetic / logic operations – the result is always stored in the accumulator.*

***The Program Counter (PC)***

* + ***This is a register that is used to control the sequencing of the execution of instructions.***
  + ***This register always holds the address of the next instruction.***
  + ***Since it holds an address, it must be 16 bits wide.***

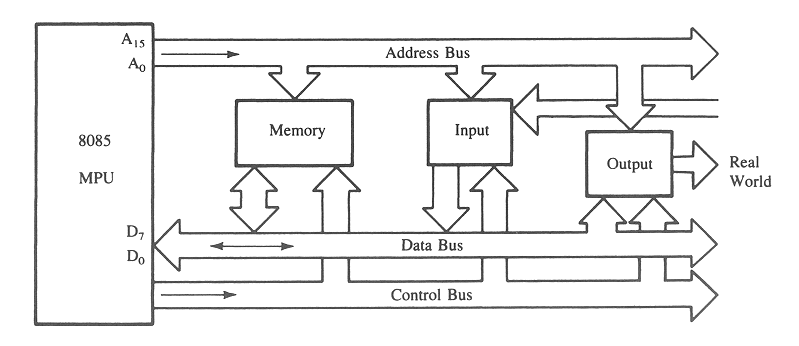
***The Stack pointer***

* + ***The stack pointer is also a 16-bit register that is used to point into memory.***
  + ***The memory this register points to is a special area called the stack.***
  + ***The stack is an area of memory used to hold data that will be retreived soon.***

***The stack is usually accessed in a Last In First Out (LIFO) fashion.***

**The Features of 8085 and Its Busses**

* ***The 8085 is an 8-bit general purpose microprocessor that can address 64K Byte of memory.***
* ***It has 40 pins and uses +5V for power. It can run at a maximum frequency of 3 MHz.***
  + ***The pins on the chip can be grouped into 6 groups:***
    - ***Address Bus.***
    - ***Data Bus.***
    - ***Control and Status Signals.***
    - ***Power supply and frequency.***
    - ***Externally Initiated Signals.***
    - ***Serial I/O ports.***
* **The 8-bit 8085 CPU or MPU – (Micro Processing Unit) communicates with the other units using a 16-bit address bus, an 8-bit data bus and a control bus.**



* ***The address bus has 8 signal lines A8 – A15 which are unidirectional.***
* ***The other 8 address bits are multiplexed (time shared) with the 8 data bits.***
  + ***So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.***
    - ***During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.***

***In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.***

**The Control and Status Signals**

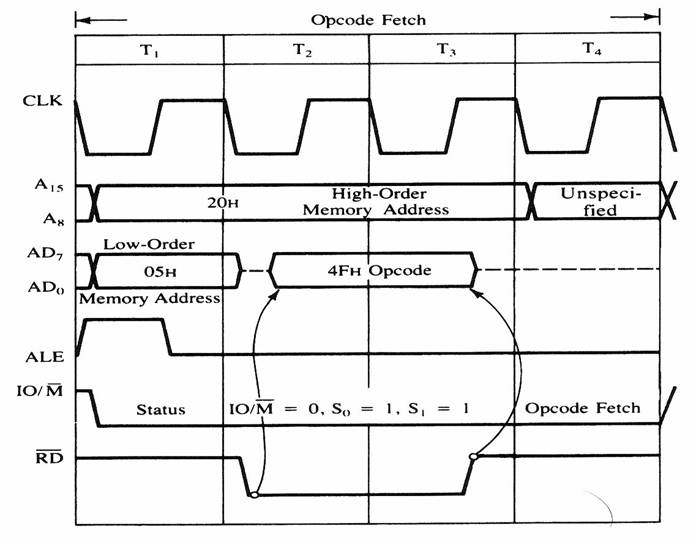
* ***There are 4 main control and status signals. These are:***
  + - ***ALE: Address Latch Enable. This signal is a pulse that become 1 when the AD0 – AD7 lines have an address on them. It becomes 0 after that. This signal can be used to enable a latch to save the address bits from the AD lines.***
    - ***RD: Read. Active low.***
    - ***WR: Write. Active low.***
    - ***IO/M: This signal specifies whether the operation is a memory operation (IO/M=0) or an I/O operation (IO/M=1).***
    - ***S1 and S0 : Status signals to specify the kind of operation being performed .Usually un-used in small systems.***

**Frequency Control Signals**

* ***There are 3 important pins in the frequency control group.***
  + ***X0 and X1 are the inputs from the crystal or clock generating circuit.***
    - ***The frequency is internally divided by 2.***
      * ***So, to run the microprocessor at 3 MHz, a clock running at 6 MHz should be connected to the X0 and X1 pins.***
  + ***CLK (OUT): An output clock pin to drive the clock of the rest of the system.***
  + ***We will discuss the rest of the control signals as we get to them.***

**Cycles and States**

* + ***T- State: One subdivision of an operation. A T-state lasts for one clock period.***
    - ***An instruction’s execution length is usually measured in a number of T-states. (clock cycles).***
  + ***Machine Cycle: The time required to complete one operation of accessing memory, I/O, or acknowledging an external request.***
    - ***This cycle may consist of 3 to 6 T-states.***
  + ***Instruction Cycle: The time required to complete the execution of an instruction.***
    - ***In the 8085, an instruction cycle may consist of 1 to 6 machine cycles.***



**Figure 4: 8085 timing diagram for Opcode fetch cycle for MOV C, A .**

**The Fetch Execute Sequence :**

* 1. The μp placed a 16 bit memory address from PC (program counter) to address bus.
     + Figure 4: at T1
       - The high order address, 20H, is placed at A15 – A8.
       - the low order address, 05H, is placed at AD7 - AD0 and ALE is active high.
       - Synchronously the IO/M is in active low condition to show it is a memory operation.
  2. At T2 the active low control signal, RD, is activated so as to activate read operation; it is to indicate that the MPU is in fetch mode operation.
  3. T3: The active low RD signal enabled the byte instruction, 4FH, to be placed on AD7 – AD0 and transferred to the MPU. While RD high, the data bus will be in high impedance mode.
  4. T4: The machine code, 4FH, will then be decoded in instruction decoder. The content of accumulator (A) will then copied into C register at time state, T4.

**The ALU**

* ***In addition to the arithmetic & logic circuits, the ALU includes the accumulator, which is part of every arithmetic & logic operation.***
* ***Also, the ALU includes a temporary register used for holding data temporarily during the execution of the operation. This temporary register is not accessible by the programmer.***

**The Flags register**

* + ***There is also the flags register whose bits are affected by the arithmetic & logic operations.***
    - ***S-sign flag*** 
      * ***The sign flag is set if bit D7 of the accumulator is set after an arithmetic or logic operation.***
    - ***Z-zero flag*** 
      * ***Set if the result of the ALU operation is 0. Otherwise is reset. This flag is affected by operations on the accumulator as well as other registers. (DCR B).***
    - ***AC-Auxiliary Carry*** 
      * ***This flag is set when a carry is generated from bit D3 and passed to D4 . This flag is used only internally for BCD operations. (Section 10.5 describes BCD addition including the DAA instruction).***
    - ***P-Parity flag***

***After an ALU operation if the result has an even # of 1’s the p-flag is set. Otherwise it is cleared. So, the flag can be used to indicate even parity.***

* + - ***CY-carry flag*** 
      * **This flag is set when a carry is generated from bit D7 after an unsigned operation.**
    - **OV-Overflow flag** 
      * **This flag is set when an overflow occurs after a signed operation.**

**BCD**

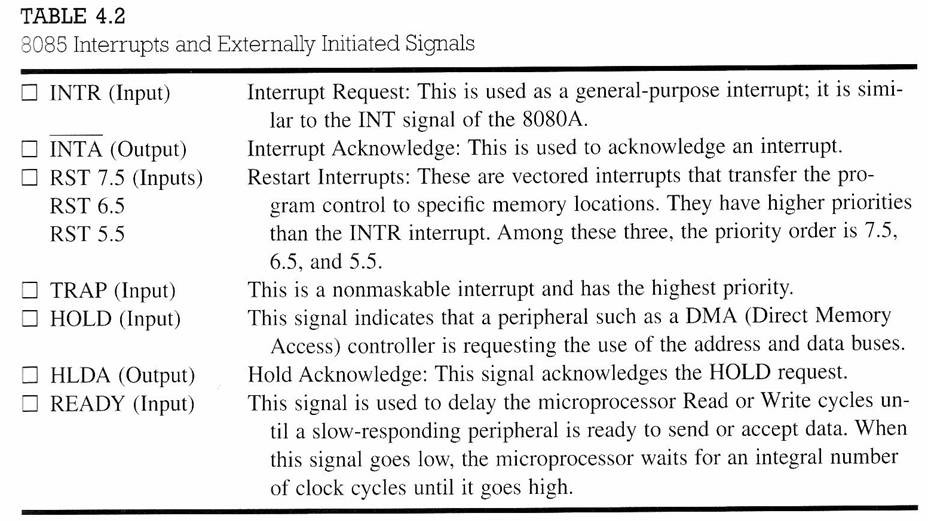
As described in the introduction, BCD takes advantage of the fact that any one decimal numeral can be represented by a four bit pattern:

As most computers store data in 8-bit [bytes](http://en.wikipedia.org/wiki/Byte), it is possible to use one of the following methods to encode a BCD number:

* **Uncompressed**: each numeral is encoded into one byte, with four bits representing the numeral and the remaining bits having no significance.
* **Packed**: two numerals are encoded into a single byte, with one numeral in the least significant [nibble](http://en.wikipedia.org/wiki/Nibble) (bits 0 through 3) and the other numeral in the most significant nibble (bits 4 through 7).
* As an example, encoding the decimal number **91** using uncompressed BCD results in the following binary pattern of two bytes:
* Decimal: 9 1 Binary : 0000 1001 0000 0001
* In packed BCD, the same number would fit into a single byte:
* Decimal: 9 1 Binary : 1001 0001
* Hence the numerical range for one uncompressed BCD byte is zero through nine inclusive, whereas the range for one packed BCD is zero through ninety-nine inclusive.
* To represent numbers larger than the range of a single byte any number of contiguous bytes may be used.  For example, to represent the decimal number **12345** in packed BCD, using [big-endian](http://en.wikipedia.org/wiki/Big-endian) format, a program would encode as follows:
* Decimal: 1 2 3 4 5 Binary : 0000 0001 0010 0011 0100 0101
* Note that the most significant nibble of the most significant byte is zero, implying that the number is in actuality **012345**.  Also note how packed BCD is more efficient in storage usage as compared to uncompressed BCD; encoding the same number in uncompressed format would consume 100 percent more storage.
* [Shifting](http://en.wikipedia.org/wiki/Logical_shift) and [masking](http://en.wikipedia.org/wiki/Mask_(computing)) operations are used to pack or unpack a packed BCD digit.  Other [logical operations](http://en.wikipedia.org/wiki/Bitwise_operation) are used to convert a numeral to its equivalent bit pattern or reverse the process.

**Interrupt Signals**

* **An interrupt is a hardware-initiated subroutine CALL.**
* **When interrupt pin is activated, an ISR will be called, interrupting the program that is currently executing.**

****

**Direct Memory Access (DMA)**

* **DMA is an IO technique where external IO device requests the use of the MPU buses.**
* **Allows external IO devices to gain high speed access to the memory.**
  + **Example of IO devices that use DMA: disk memory system.**
* **HOLD and HLDA are used for DMA.**
* **If HOLD=1, 8085 will place it address, data and control pins at their high-impedance.**
* **A DMA acknowledgement is signaled by HLDA=1.**

**The 8085 machine cycles**

* ***The 8085 executes several types of instructions with each requiring a different number of operations of different types. However, the operations can be grouped into a small set.***
* ***The three main types are:***
  + - ***Memory Read and Write.***
    - ***I/O Read and Write.***
    - ***Request Acknowledge.***
* ***These can be further divided into various operations (machine cycles).***

**Op-code Fetch Machine Cycle**

* ***The first step of executing any instruction is the Op-code fetch cycle.***
  + ***In this cycle, the microprocessor brings in the instruction’s Op-code from memory.*** 
    - ***To differentiate this machine cycle from the very similar “memory read” cycle, the control & status signals are set as follows:***
      * ***IO/M=0, s0 and s1 are both 1.***
  + ***This machine cycle has four T-states.***
    - ***The 8085 uses the first 3 T-states to fetch the opcode.***
    - ***T4 is used to decode and execute it.***
  + ***It is also possible for an instruction to have 6 T-states in an opcode fetch machine cycle.***

**Memory Read Machine Cycle**

* ***The memory read machine cycle is exactly the same as the opcode fetch except:***
  + ***It only has 3 T-states***
  + ***The s0 signal is set to 0 instead.***

***The Memory Read Machine Cycle***

* + ***To understand the memory read machine cycle, let’s study the execution of the following instruction:***
    - ***MVI A, 32***
  + ***In memory, this instruction looks like:***
    - ***The first byte 3EH represents the opcode for loading a byte into the accumulator (MVI A), the second byte is the data to be loaded.***
  + ***The 8085 needs to read these two bytes from memory before it can execute the instruction. Therefore, it will need at least two machine cycles.*** 
    - * ***The first machine cycle is the opcode fetch discussed earlier.***
      * ***The second machine cycle is the Memory Read Cycle.***

**Machine Cycles vs. Number of bytes in the instruction**

* ***Machine cycles and instruction length, do not have a direct relationship.*** 
  + ***To illustrate lets look at the machine cycles needed to execute the following instruction.***
    - ***STA 2065H***
    - ***This is a 3-byte instruction requiring 4 machine cycles and 13 T-states.***
    - ***The machine code will be stored   
      in memory as shown to the right***
    - ***This instruction requires the following 4 machine cycles:***
      * ***Op-code fetch to fetch the op-code (32H) from location 2010H, decode it and determine that 2 more bytes are needed (4 T-states).***
      * ***Memory read to read the low order byte of the address (65H) (3 T-states).***
      * ***Memory read to read the high order byte of the address (20H) (3 T-states).***
      * ***A memory write to write the contents of the accumulator into the memory location.***

**The Memory Write Operation**

* ***In a memory write operation:***
  + ***The 8085 places the address (2065H) on the address bus***
  + ***Identifies the operation as a memory write (IO/M=0, s1=0, s0=1).***
  + ***Places the contents of the accumulator on the data bus and asserts the signal WR.***
  + ***During the last T-state, the contents of the data bus are saved into the memory location.***