Clock Cycle

- The speed of a computer microprocessor or CPU, is determined by the clock cycle, which is the amount of time between two pulses of an oscillator.
- Generally speaking, the higher number of pulses per second, the faster the computer processor will be able to process information. The clock speed is measured in Hz
- typically either megahertz (MHz) or gigahertz (GHz).
- Computer microprocessors can execute one or more instructions per clock cycle, depending on the type of microprocessor.
- Early computer microprocessors and slower microprocessors can only execute one instruction per clock cycle but faster more advanced microprocessors can execute multiple instructions per clock cycle, processing data more efficiently.

Instructions of a Microprocessor

- There is some definite things required for the execution of an instruction like the addressing modes, instruction set, operation codes etc.

Addressing modes

- The term addressing modes refers to the way in which the operand of an instruction is specified.
- Information contained in the instruction code is the value of the operand or the address of the result/operand.

Instruction format

- The set of simple tasks that the processor can perform are called the instruction set.
- instruction set is usually composed of two parts: the first part is a mnemonic called the OPCODE, which tells what the microprocessor is supposed to do.
- The second part, which may be composed of one or two words, contains either data or addresses where data manipulation is to take place.
- The actual values of these words depend upon the OPCODE. Usually, the data which each OPCODE operates on is called operand(s). The entire instruction set and its syntactic rules form a grammar that the CPU can understand and act upon. It is called assembly language.

Mnemonic

The word MNEMONIC means "A device such as a pattern of letters, ideas, or associations that assists in remembering something". So, it is usually used by assembly language programmers to remember the "OPERATIONS" a machine can do, like "ADD" and "MUL" and "MOV" etc.

Op code (operation code)

The OPCODE is part of an instruction word that is interpreted by the processor as representing the operation to perform, such as read, write, jump, add etc. Many instructions will also have OPERANDS that affect how the instruction performs, such as saying from where in memory to read or write, or where to jump.

Microprocessors According to Characteristics

- RISC processors
- CISC processors
- Special processors.

RISC processor

- RISC stands for **'Reduced instruction set computer'**. This microprocessor was designed to perform a smaller number of types of computer instructions so that it can operate at a higher speed. Characteristics of RISC processors are given below. Relatively few instructions and addressing modes

- Memory access limited to load and store instructions.
- All operations done within the registers of CPU
- Fixed length, easily decoded instruction format
- Single cycle instruction execution i.e. required only one cycle to execute a definite instruction.
- The microprocessor is designed using hardwired control.
- It has several general purpose registers. Some popular RISC processors are:
  - ✓ PA-RISC: HP700LC
  - ✓ POWER PC- 601,604,615,620

Intel's processors of x86 Families are the best example of CISC processors. We can see that the RISC and CISC processors can easily be differentiated by their own characteristics. Besides pipelining and superscalar architectures are the base methods to design RISC which is not true for CISC processors.

CISC Processors

- CISC Stands for "**Complex Instruction Set Computing.**" This is a type of microprocessor design. The CISC architecture contains a large set of computer instructions that range from very simple to very complex and specialized. The design was intended to compute complex instructions in the most efficient way.
- Some characteristics of CISC processors are given below
  - Large number of addressing modes and instructions
  - Some instructions perform special task
  - Variable length instruction format
  - Instructions that manipulate control
  - Several cycles may be required to execute one instruction.
  - The microprocessor is designed using code control.
  - It has small number of general purpose registers
- Intel's processors of x86 Families are the best example of CISC processors. We can see that the RISC and CISC processors can easily be differentiated by their own characteristics. Besides pipelining and superscalar architectures are the base methods to design RISC which is not true for CISC processors.

Special processors

- There are several other processors, which are useful in special purposes. Some of these processors are briefly discussed below.

- Coprocessor: Coprocessor is very similar to general purpose microprocessor.

It is designed for a specific function. But coprocessor can handle its particular function many times faster than the ordinary general purpose microprocessor. The most well known coprocessor is math-coprocessor.

- Input/output processor: This is one of the most important microprocessors. With this processor a large set of I/O device can be controlled with minimum CPU involvement. The I/O processors take care of most of the tasks involved in controlling the terminals. The common examples of I/O processors are keyboard/mouse controller, graphic display controller etc.

- Transputer (Transistor computer) It is a high performance microprocessor designed to facilitate inter process and inter processor communications and targeted at the efficient exploitation of very large scale integration. The most important feature of transputer is it external links, which enables it to be used as a building block in the construction of low cost, high performance multiprocessing system.

Digital Signal Processor (DSP)

- This processor is specially designed to handle real world analog signals that have been converted into digital representation.
- Modem function, speech recognition 2D and 3D graphics acceleration audio and video compression etc are the applications of DSP.

.

Assembly Language Programming

Assembly language programming is writing machine instructions in mnemonic form, using an assembler to convert these mnemonics into actual processor instructions and associated data.

- An assembly language is a low-level programming language for microprocessors and other programmable devices.

Features of Assembly Language Programming

- Assembly language is the most basic programming language available for any processor.
- Assembly languages generally lack high-level conveniences such as variables and functions.
- It has the same structures and set of commands as machine language, but it allows a programmer to use names instead of numbers.
- This language is still useful for programmers when speed is necessary or when they need to carry out an operation that is not possible in high-level languages.
- Some important features of assembly language programming are given below:
- Allows the programmer to use mnemonics when writing source code programs, like 'ADD' (addition), 'SUB' (subtraction), JMP (jump) etc
- Variables are represented by symbolic names, not as memory locations, like MOV A, here 'A' is the variable.
- Symbolic code
- error checking
- Changes can be quickly and easily incorporated with a re-assembly
- Programming aids are included for relocation and expression evaluation.

- Variables are represented by symbolic names, not as memory locations, like MOV A, here ‘A’ is the variable.
- Symbolic code
- error checking
- Changes can be quickly and easily incorporated with a re-assembly
- Programming aids are included for relocation and expression evaluation.

Advantages of assembly language programming

- Easy to understand and use
- Easy to locate and correct errors
- Easy to modify
- No worry to address.
- Efficient than machine language programming

Disadvantages of assembly language programming

- The programmer requires knowledge of the processor architecture and instruction set.
- Machine language coding

  Many instructions are required to achieve small tasks
- Source programs tend to be large and difficult to follow

Instructions set of Intel 8085 microprocessor according to assembly language programming

- ADD r:- add
- ADD M:- Add to Memory
- ADC r:-add with carry
- ADC M:-add with carry to memory
- LDI :- This instruction stands for load immediate and it means load' (put) into a particular named register a certain value named in the instruction
- LDA a:- load accumulator direct
- SLA: -means shift the contents of a register one place to the left.
- MOV:-means to move data from one place to desired place
- MOV M A:- means to move the result from memory to register A
- STA:- Store on register(accumulator or others)
- STA a: - store to accumulator.
- JMP- Jump to somewhere
- SUB r :– subtract
- SUB M:- subtract memory
- CY:-carry

Assembly Language Basic Syntax

An assembly program can be divided into three sections:

- The data section
- The bss section
- The text section

The data Section :The data section is used for declaring initialized data or constants. This data does not change at runtime. You can declare various constant values, file names or buffer size etc. in this section.The syntax for declaring data section is:

```
section .data
```

The bss Section: The bss section is used for declaring variables. The syntax for declaring bss section is:

```
section .bss
```

The text section: The text section is used for keeping the actual code. This section must begin with the declaration global main, which tells the kernel where the program execution begins.

The syntax for declaring text section is:

```
section .text
 global main
main:
```

Comments

Assembly language comment begins with a semicolon (;). It may contain any printable character including blank.

It can appear on a line by itself, like:

```
; This program displays a message on screen
```

or, on the same line along with an instruction, like:

```
add eax ,ebx ; adds ebx to eax
```

Assembly Language Statements

Assembly Language programs consist of three types of statements:

- Executable instructions or instructions
- Assembler directives or pseudo-ops
- Macros

- The executable instructions or simply instructions tell the processor what to do. Each instruction consists of an operation code (opcode). Each executable instruction generates one machine language instruction.
- The assembler directives or pseudo-ops tell the assembler about the various aspects of the assembly process.
- These are non-executable and do not generate machine language instructions.
- Macros are basically a text substitution mechanism.

Syntax of Assembly Language Statements

Assembly language statements are entered one statement per line. Each statement follows the following format:

```
[label] mnemonic [operands] [;comment]
```

- The fields in the square brackets are optional. A basic instruction has two parts, the first one is the name of the instruction (or the mnemonic) which is to be executed, and the second are the operands or the parameters of the command. The following are some examples of typical assembly language statements:

```
INC COUNT          ; Increment the memory variable COUNT
MOV TOTAL, 48      ; Transfer the value 48 in the
                   ; memory variable TOTAL
ADD AH, BH         ; Add the content of the
                   ; BH register into the AH register
AND MASK1, 128     ; Perform AND operation on the
                   ; variable MASK1 and 128
ADD MARKS, 10      ; Add 10 to the variable MARKS
MOV AL, 10         ; Transfer the value 10 to the AL register
```

The Hello World Program in Assembly

The following assembly language code displays the string 'Hello World' on the screen:

```
INC COUNT                                   ; Increment the memory variable COUNT
section .text
 global main ;must be declared for linker (ld)
main:                                       ;tells linker entry point
 mov edx,len                                ;message length
 mov ecx,msg                                ;message to write
 mov ebx,1                                  ;file descriptor (stdout)
 mov eax,4                                  ;system call number (sys_write)
 int 0x80                                   ;call kernel
mov eax,1                                   ;system call number (sys_exit)
 int 0x80                                   ;call kernel
section .data
msg db 'Hello, world!', 0xa                 ;our dear string
len equ $ - msg                             ;length of our dear string
```

When the above code is compiled and executed, it produces following result;

```
Hello, world!
```