1. What is the 68000 Processor? (mentioning about 68000 addressing modes: inherent, immediate, relative, extended, indexed)?

Motorola 68000 (MC68000) is the first member of 680x0 line of microprocessors. Internally the 68000 is a 32-bit microprocessor - it has 32-bit data and address registers. Externally the processor has 16-bit data bus and 24-bit address bus, which limits the size of addressable memory to 16 MB.

addressing modes:

●Data register direct mode

●Address register direct mode

●Address register indirect mode

●Address Register Indirect with Postincrement mode

●Address Register Indirect with Predecrement mode

●Address Register Indirect with Displacement mode

● Address Register Indirect with Index mode

●Memory Indirect Post-indexed mode

●Memory Indirect Pre-indexed mode

●Program Counter Indirect with Displacement mode

●Program Counter Indirect with Index modes

●Program Counter Memory Indirect modes

●Absolute addressing modes

1. What is the 68000 Assembly Language?

In computer programming, assembly language (or assembler language),[1] often abbreviated asm, is any low-level programming language in which there is a very strong correspondence between the instructions in the language and the architecture's machine code instructions.[2] Because assembly depends on the machine code instructions, every assembler has its own assembly language which is designed for exactly one specific computer architecture. Assembly language may also be called symbolic machine code.

68000 Assmbly Language is the assembly language for Motorola 68000 .

1. Why we are using Assemblers?

Some assemblers may also be able to perform some simple types of [instruction set](https://en.wikipedia.org/wiki/Instruction_set" \o "Instruction set)-specific [optimizations](https://en.wikipedia.org/wiki/Compiler_optimization" \o "Compiler optimization). One concrete example of this may be the ubiquitous [x86](https://en.wikipedia.org/wiki/X86" \o "X86) assemblers from various vendors. Most of them are able to perform jump-instruction replacements (long jumps replaced by short or relative jumps) in any number of passes, on request. Others may even do simple rearrangement or insertion of instructions, such as some assemblers for [RISC](https://en.wikipedia.org/wiki/RISC" \o "RISC) [architectures](https://en.wikipedia.org/wiki/Instruction_set_architecture" \o "Instruction set architecture) that can help optimize a sensible [instruction scheduling](https://en.wikipedia.org/wiki/Instruction_scheduling" \o "Instruction scheduling) to exploit the [CPU pipeline](https://en.wikipedia.org/wiki/CPU_pipeline" \o "CPU pipeline) as efficiently as possible.[*[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)*]

A [programmer](https://www.techopedia.com/definition/4813/programmer) can still gain a lot if he/she can learn to code in assembly language and implement it. These days, assembly language makes it possible to manipulate hardware directly, address critical issues concerning performance and also provide access to special instructions for processors. Uses of assembly language include coding device drivers, real-time systems, low-level embedded systems, boot codes, reverse engineering and more.

1. What is 68000 Simulator?

The Crossware S68000NT Simulator creates a virtual 68000 microprocessor that runs on your PC. It allows you to run your 68000 program without any 68000 hardware and watch your code execute in the Windows graphic environment.

**HIGHLIGHTS**

* Full source level debugging
* Seamless integration with the Embedded Development Studio
* Dockable windows
* Multiple watch windows for local and global variables
* Views of all memory areas, registers and disassembled program
* State capture to capture and restore the complete internal state of the microcontroller
* Source code profiling
* Code and data coverage analysis
* Multiple cycle counters

**TECHNICAL DETAILS:**

●**Seamless Integration with the Embedded Development Studio**

●**Full Source Level Debugging**

●**Multiple Watch Windows for Local and Global Variables**

●**Call Stack**

●**Views of Memory Areas**

●**View Named Registers**

●**Exception Simulation and Views**

●**State Capture**

●**Source Code Profiling**

●**Code and Data Coverage**

●**Multiple Cycle Counters**

●**Memory**

●**Program File Formats**

●**Availability**

**Preference:**

Hamacher, Vranesic & Zaky. *Computer Organization McGraw-Hill Science August* 2, 2001

# Thomas, P. Skinner . *Assembly Language 68000 (PWS Series in Engineering) 2nd (second) Revised Edition,1993*

<https://en.wikipedia.org/wiki/Motorola_68000>

<https://en.wikipedia.org/wiki/Assembly_language>