Assembly 1 Project

What is the 68000 processor? Motorola 68000 type central processing unit, or MC68000, 680x0, m68000, m68k, 68k; is a 16/32 Bit CISC (Complex Instruction Set) microprocessor, produced by the semiconductor division of Motorola (now independently become Freescale). The MC68000 hit the market in 1979 as the first member of the M68K processor family. Due to the internal use of 32-bit buses and registers, it is basically compatible with subsequent pure 32-bit products at the software level (instruction set). (Motorola 68000, n.d.)[1]

About 68000 Standard addressing mode. The 68000 offers several addressing modes, collectively referred to as Effective Addressing (EA). In the CPU reference manual, there are often notations. This means that one (but usually not all) effective addressing can be used on each of the destination and source operands. (Kane, Hawkins, & Leventhal)[2]

The following two modes do not involve memory at all:

1. Inherent addressing means that the operation code alone tells the processor what to do.

2. Register addressing means that the operand is contained in a register.

Common addressing modes that involve memory are as follows:

3. Immediate addressing means that operand is located immediately after the operation code in program memory.

4. Direct addressing means that the address to be used follows the operation code in program memory.

5. Index addressing means that the address to be used is the sum of a base address and an index or offset.

6. Indirect addressing means that the address to be used is either in a register or in memory. That is, the instruction tells the processor where the address is, not where the data is. 7. Relative addressing means that the operand is located a certain distance from the current position of the program.

What is the 68000 Assembly Language? like any other assembly language, is composed of two types of statements: the assembler directive and the executable instruction. An executable instruction is one of the processor's valid instructions which is translated by the Assembler into machine language and executed by the CPU. An assembler directive, on the other hand, is just an indication to the Assembler about the program and its environment. Assembler directives are not translated into machine language. (9.3 ELEMENTARY PROGRAMMING IN 68000 LANGUAGE, n.d.) [3]

Why are we using Assemblers? An assembler tries to map one-to-one the machine mnemonics to the underlying binary opcodes. As such it is the most expressive language to a particular machine. Some languages attempt to hide 'pointers' or memory addresses. All languages hide register allocation and mapping variables to either stack slots or physical registers. The job of the optimizing compiler is to map the high-level language to the underlying machine language. The algorithms used can be quite exhaustive as a computer can search a large number of solutions faster than a human and find an optimal solution. (Why do we even need assembler when we have compiler?, n.d.)[4]

Programming in assembly language allows the control of every aspect of the computer

hardware. Many applications require procedures that are either impossible or inefficient

with computer languages such as BASIC. You may be a professional computer user who

has a need for a laboratory control computer, such real time applications often require

some assembly language programming. Regardless of your reason for learning assembly

language, it is challenging and rewarding when your programs start to run.

What is 68000 Simulator? The Motorola 68000 simulator included in the BSVC distribution simulates the 68000 at the software level. This means the simulator does not understand what goes on in the 68000 at the hardware level. Instead, the simulator performs a set of actions for each instruction that gives the same result. (Chapter 3 Motorola 68000 Simulator and Assembler, n.d.)[5]

References:

1. Motorola\_68ooo page from Wikipedia

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

1. 68000 assembly language programming section 3-7 [68000\_assembly\_language\_programming.pdf](https://datassette.nyc3.cdn.digitaloceanspaces.com/livros/68000_assembly_language_programming.pdf)
2. Elementary programming in 68000 languages

[9.3 ELEMENTARY PROGRAMMING IN 68000 LANGUAGES](https://www.cs.mcgill.ca/~cs573/fall2002/notes/lec273/lecture9/)

# [Why do we even need assembler when we have compiler?](https://stackoverflow.com/questions/51780158/why-do-we-even-need-assembler-when-we-have-compiler)

[Why do we even need assembler when we have compiler?](https://stackoverflow.com/questions/51780158/why-do-we-even-need-assembler-when-we-have-compiler)

1. Motorola 68000 Simulator and Assembler

[Chapter 3 Motorola 68000 Simulator and Assembler](https://www.csc.kth.se/hacks/doc/bsvc/chap3.htm)