68000 Processor: Addressing Modes, Assemblers & Simulators

Sanam Jena

Stevens Institute of Technology

Author Note

Sanam Sritam Jena, Student at Stevens Institute of Technology

This research project is being presented as a part of evaluation & fulfilment of the curriculum of CS-550 Computer Organization & Programming under the guidance of Dr. Edward Banduk.

Correspondence concerning to this article should be addressed to Sanam Jena, Stevens Institute of Technology, Hoboken, NJ 07030.

Contact: sjena@stevens.edu

Abstract

The Motorola 68000 is a 16/32-bit CISC microprocessor implementing 32-piece instruction set, with 32-piece registers and a 32-piece internal data bus. The primary concept in computing in both high-level language and low-level language is the addressing mode. Computers do operations on data and we have to specify the source of the data. The different ways to mention the source or destination of an operand is called addressing modes. The various types of addressing modes are immediate, inherent, relative, extended & indexed are the ones discussed in this paper. Further discussing about assembly languages, it is defined as, a low-level programming language for a computer, or other programmable gadget. Assemblers are still being used since people work more comfortably with high-level languages, so assemblers help people to interpret these high-level languages to machine language. The Motorola 68000 simulator incorporated into the BSVC distribution simulator simulates the 68000 at the software level.

Keywords: CISC, ALU, BUS, edition, APA, Effective Address, Program Counter, IMM, BSVC, UART, RAM

68000 Processor: Addressing Modes, Assemblers & Simulators

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

The **Motorola 68000** ("'sixty-8,000"'; additionally, called the **m68k** or **Motorola 68k**, "sixty-eight-kay") is a 16/32-piece CISC chip, presented in 1979 by Motorola Semiconductor Products Sector.

The structure actualizes a 32-piece instruction set, with 32-piece registers and a 32-piece internal data bus. The address bus is 24-bits and doesn't utilize memory segmentation, which made it well known with software engineers. Inside, it utilizes a 16-piece data ALU and two extra 16-piece ALUs utilized generally for addresses and has a 16-piece external data bus. For this explanation, Motorola alluded to it as a 16/32-piece processor.[2]

As one of the primary generally accessible processors with a 32-bit instruction set, and running at comparatively high speeds for the time, the 68k was a mainstream plan through the 1980s. It was generally utilized in new generation of PCs with graphical UIs, including the Apple Macintosh, Commodore Amiga, Atari ST and numerous others. It competed mainly against the Intel 8088, found in the IBM PC, which it effectively beat. The 68k and 8088 pushed different designs, similar to the Zilog Z8000 and National Semiconductor 32016, into specialty showcases, and made Motorola a noteworthy player in the CPU space.[1]

The 68k was before long extended with extra relatives, executing full 32-bit ALUs as a major aspect of the developing Motorola 68000 arrangement. The first 68k is for the most part

programming forward-compatible with the remainder of the line in spite of being constrained to a 16-piece wide outer transport. The primary concept in computing in both high-level language and low-level language is the addressing mode. Computers do operations on data and we have to specify the source of the data. The different ways to mention the source or destination of an operand is called addressing modes.

68000 processor has many addressing modes, First, in **Immediate Addressing Mode**, the operand is a real value instead of a reference to a memory address. The 68000 assembler does prompt work by prefixing the operand with the '#' symbol.[3] Example ADD #7, D4 means to add the value 7 in the register D4 and after that to store the result in the register D4. Upon comparing with absolute addressing method, immediate addressing method allows the user to specify a constant instead of a variable, absolute addressing gives you a chance to point to an operand and then the computer needs to access the memory or the register to get the result stored. Second, **index mode** is separated into two parts: **Basic index mode** and **full index mode**, discussing about the basic index mode, effective address is created by increasing the value of the content register with a constant (the constant comprises of 16 bits). Example Add -$50(A3), D1 means to deduct $50 from A3 and the value present in D1 and store the same result in D1. In full index mode, we add a constant value to the index of both the registers (the constant has 8 bits only). Example, SUB 7(A2, D1), D2 means Effective Address (EA)= 7 + (A2) + [D1], [D2] – [EA] → D2. Third, **relative mode:** Effective Address (EA) is produced by increasing the value of content of the Program Counter (PC), equivalent to the index mode with the only difference is here we add a value to the PC, not the address register, and in respect to the current instruction in the program, Motorola 68000 comprises of two relative modes: a) Basic relative mode: ADD -$100(PC),D2 means reduce the PC by $100and then add (D2), and store the outcome in (D2). B)

Full relative mode: SUB 5(PC, D0), D1 means [D1] – [[PC] + [D0] + 5] → D1 [4]. The **extended addressing mode** is recognized and distinguished from the rest of the addressing modes by the presence of a 16-bit address of the 2nd operand, and the nonappearance of the # symbol to distinguish it from the IMM mode. It allows processing of variables in any area of the memory space, as the 2nd operand is specified by a 16-bit address. Example, A ← A + ($0101), this includes the contents present in the memory location $0101 to A. The kind of instruction with no operation and no operand means it is of type **Inherent Addressing Mode (INH)**[5]. Example, lets take into consideration that the instruction is stored at $1010, to increase register A by1, we perform (A) + $01 → A INH 42

2) What is the 68000 Assembly Language?

Assembly Language is a low-level programming language for a computer, or other programmable gadget. Every assembly language is explicit to a specific computer architecture. 68000 Assembly Language is utilized to make Motorola 68000 CPU work, it is changed over into executable machine code by a utility program alluded to as an assembler (the user input the 68000 assembly language, then it is converted). The assembly language utilizes a memory aid to speak to each low-level machine instruction, ordinarily each architectural register.

3) Why are we using Assemblers?

In spite of the fact that individuals can program with assembly language, in any case, these days we for the most part compose programs in high-level language. Since people work more comfortably with high-level languages, so assemblers help people to interpret these high-level languages to machine language. An assembler program generates object code by converting

combinations memory and syntax of instructions and addressing modes into their numerical equivalent which is easy for the machine to understand and perform. This is the primary reason why humans tend to avoid using assembly language.

4) What is the 68000 Simulator?

The Motorola 68000 simulator incorporated into the BSVC distribution simulator simulates the 68000 at the software level. Means the simulator doesn't comprehend what goes on in the 68000 at the equipment level. Rather, the simulator does a set of predefined activity for every instruction that gives the equivalent result. The simulator allows two devices that can be connected to the chip. These devices are the M68681 Dual UART and RAM.[6]

Reference

[1] *History of 68000* Retrieved from https://en.wikipedia.org/wiki/Motorola_68000

[2]Phoenix, AZ: Motorola. *Motorola M68000 Family Programmer's Reference Manual.* 1992. P.1-1.

[3]Clements, A *68K Addressing Modes* Retrieved from http://alanclements.org/68kaddressingmodes3.html

[4]Yuan. F. Zheng , *68000 Addressing Modes* Retrieved from http://www2.ece.ohio-state.edu/~zheng/ece5362/lecture-notes/68000-Addressing-Modes.pdf

[5] *Addressing mode* Retrieve from http://ece.eng.umanitoba.ca/undergraduate/ECE3610/LectureNotes/Lecture%207%20AM.pdf

[6]Bradford W. Mott *Motorola 68000 Simulator and Assembler* Retrieved from https://www.nada.kth.se/hacks/doc/bsvc/chap3.htm