**Assembly 1 Project**

Pawan Kumar

Department of Computer Science, Stevens Institute of Technology

CPE 550-A, Computer Organization and Programming

Dr. Edward Banduk

October 4th, 2021

**Assembly 1 Project**

Q1) What is the 68000 Processor? (mentioning about 68000 addressing modes: inherent, immediate, relative, indexed).

The Motorola 68000 (sixty-eight-thousand; also called m68k, Motorola 68k, sixty-eight-kay) is a 16/32-bit complex instruction set computer (CISC) microprocessor, introduced in 1979 by Motorola Semiconductor Products Sector.

The design implements a 32-bit instruction set, with 32-bit registers and a 16-bit internal data bus. The address bus is 24 bits and does not use memory segmentation, which made it easier to program for. Internally, it uses a 16-bit data arithmetic logic unit (ALU) and two more 16-bit ALUs used mostly for addresses, and has a 16-bit external data bus. For this reason, Motorola termed it a 16/32-bit processor.

As one of the first widely available processors with a 32-bit instruction set, and running at relatively high speeds for the era, the 68k was a popular design through the 1980s. It was widely used in a new generation of personal computers with graphical user interfaces, including the Macintosh, Amiga, Atari ST, and X68000. The 1988 Mega Drive console is also powered by a 68000.

The 68k was soon expanded with more family members, implementing full 32-bit ALUs as part of the growing Motorola 68000 series. The original 68k is generally software forward-compatible with the rest of the line despite being limited to a 16-bit wide external bus.



Figure1: Pre-release XC68000 chip made in 1979

After 42 years in production, the 68000 architecture is still in use.

Addressing Modes:

Immediate addressing –

The MC68000 has byte, word and long word immediate instructions. Byte and Word immediate instructions provide the immediate operand in the word immediately following the instruction’s object code. For byte immediate instructions, the immediate data is in the low-order byte of the word immediately following the instruction’s object code.

In the standard MC68000 assembly language, we specify immediate addressing by preceding the operand with the # symbol. For example, the MC68000 assembler converts the statement

ADD #$1066.D3

(# sign means “immediate addressing” and

$ sign means “hexadecimal”)

An ADD instruction that adds the value 106616 to data register D3.

Inherent Addressing-

Motorola literature includes the instructions, which use inherent addressing within a category of addressing which they describe as implicit. This category includes Branch instructions which always affect the program counter, certain Move instructions which affect specific registers such as status register or stack pointer, and jump instruction, which always affect the program counter. These instructions are not however, inherent addressing in the strict sense since they allow or demand additional addressing information; the operation codes for these instructions are not complete by themselves.

Relative Addressing-

Effective address (EA) is generated by adding a constant value to the content of the Program Counter

• Same as index mode except constant added to PC, not an (address register)

• Relative to current instruction in the program

• 68000 has two relative modes

1. Basic relative mode: ADD -$100(PC), D2; [[PC]-$100] +[D2] → D2
2. Full relative mode: SUB 5(PC, D0), D1 [D1] - [[PC] + [D0] + 5] → D1

Index Addressing-

a) Basic index mode

Effective address (EA) is generated by adding a constant value to the content

of an address register (the constant has 16 bits).

e. g. ADD -$100(A4), D2;

[[A4] - $100] + [D2] → D2

b) Full index mode

Effective address (EA) is generated by adding a constant value to the contents

of two registers (the constant has 8 bits only).

e. g. SUB 5(A3, D0), D1;

EA = 5 + [A3] + [D0], [D1] - [EA] → D1

Q2) What is the 68000 Assembly Language?

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 actually 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.

Executable instructions can be divided into several categories:

* data movement instructions
* integer arithmetic instructions
* logic instructions
* shifts and rotations
* bit operations
* comparisons
* program control instructions

Q3) Why we are using Assemblers?

The assembler program translates a user program, or “source” program written with mnemonics, into a machine language program, or “object” program, which the microcomputer can execute. The assembler’s input is a source program and its output is an object program. Early assemblers did little more than translate the mnemonic names of instructions and registers into their binary equivalents. However, most assemblers now provide such additional features as:

• Allowing the user to assign names to memory locations, input and output devices, and even sequences of instructions

• Converting data or addresses from various number systems (for example, Introduction to Assembly Language Programming 1-7 decimal or hexadecimal) to binary and converting characters into their ASCII or EBCDIC binary codes

• Performing some arithmetic as part of the assembly process

• Telling the loader program where in memory parts of the program or data should be placed

Q4) What is the 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.

The 68000 simulator provides two devices that can be attached to the microprocessor. These devices are the M68681 Dual UART and RAM (the Windows 95 version does not support the DUART at this time). Each of these devices is described below.

M68681 Dual UART Device

The M68681 Dual Universal Asynchronous Receiver/Transmitter (DUART) device is modelled after the Motorola MC68681 DUART. The dialog box shown in Figure allows you to customize the M68681 device when it is attached to the 68000.

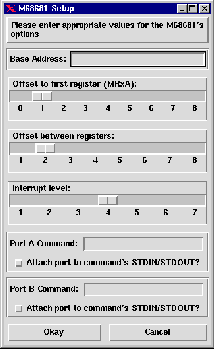


Figure2: M68681 Device Setup Dialog

Using this dialog, you can select the:

* Base address of the device
* Byte offset to the first register of the device
* Offset between registers
* Interrupt level
* Port commands
* Port input and output attachments

Features

The M68681 device supports the following features:

* Two, independent, full-duplex asynchronous Receiver/Transmitter ports
* Each port can be connected to a Unix process
* Independently programmable baud rate for each Receiver and Transmitter
  + 18 Fixed rates: 50 to 38400 baud
* Programmable data format allowing five to eight data bits
* Programmable channel modes
  + Normal (full-duplex)
  + Automatic echo
* Versatile interrupt system
  + Single interrupt output with four maskable interrupting conditions
  + Interrupt vector output on interrupt acknowledge

RAM Device

The RAM device allows you to attach a specified amount of memory to the 68000. When the RAM device is attached to the simulator, using the Edit Setup option, the dialog shown in Figure 3-2 is displayed. This dialog allows you to enter the base address and size of the RAM module. Both of these values are in hexadecimal.

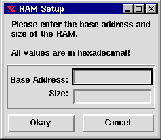


Figure3: RAM Device Setup Dialog

**References**

1. Wikimedia Foundation. (2021, August 31). Motorola 68000. Wikipedia. Retrieved October 1, 2021, from https://en.wikipedia.org/wiki/Motorola\_68000.
2. Chapter 3 Motorola 68000 simulator and Assembler. chap3.htm. (n.d.). Retrieved October 1, 2021, from https://www.csc.kth.se/hacks/doc/bsvc/chap3.htm
3. Savadjiev, P. (n.d.). Lecture 9: The registers and the memory of the 68000; intro to 68000 programs. Retrieved October 1, 2021, from https://www.cs.mcgill.ca/~cs573/fall2002/notes/lec273/lecture9/.
4. Kane, G., Hawkins, D., & Leventhal, L. A. (1981). Chapter 1,2,3. In 68000 assembly language programming (pp. 1–1-3–31). essay, Osborne/McGraw-Hill.