# AK68: A New Cross-Assembler and Integrated CPU Simulator For The Motorola M68000 Microprocessor

Dr. Sohail Anwar, Durel Hauser, Roy Sheehan
Penn State Altoona/US Postal Service/Sheehan Electronics

## ABSTRACT

This paper describes a cross-assembler and integrated CPU simulator for the Motorola M68000 microprocessor developed by the authors for use in an undergraduate introductory course on microprocessors. The principal design objective of the authors was to create a less intrusive or more user friendly assembler combined with a dynamic model of program execution to encourage deductive problem solving in place of undesirable trial and error activity.

The design of the cross-assembler centered on three goals:
(1) The assembler must discriminate between labels, operators, and operands without any strict set of identification rules.
(2) The assembler must produce a broad set of error messages which provide corrective guidance as opposed to a vague error identification. That is, the error messages must distinguish assembler errors from the Motorola M68000 assembly language syntax errors, and the assembler must graphically highlight the line element (label, operator, or operand) which caused the error. Furthermore, the error message must indicate, as possible, the needed corrective action.
(3) The assembler must include a mouse operated “pop up” hexadecimal/binary/decimal calculator.

The design goals listed above required two main views of this cross-assembler/simulator: (i) an exterior, or program flow view and (ii) an interior or CPU register view. Two selectable windows were created to fulfill this requirement. The assembler’s output (a hexadecimal object code listing) is piped to the assembler’s text editor where the line addressed by the simulated CPU program counter instruction fetch is highlighted. The interior CPU view contains all of the Motorola M68000 internal registers, stack pointers, and the status register.

The Motorola M68000 simulator does not emulate an internal multi-phase CPU clock. Multiple simultaneous exceptions (more than one exception occurring during the execution of a single instruction) result in exception processing which may or may not be equivalent to the order of execution which would occur in a real Motorola M68000 hardware environment. For this reason, when simultaneous exception errors are detected, the program halts and a message is displayed. During two years of use at Penn State Altoona, this limitation has never been observed to restrict the primary utility of the program, that is, to visually illustrate the operation of the Motorola M68000 microprocessor.

This paper provides a detailed description of the cross-assembler/integrated CPU simulator titled AK68. Design considerations are described and details of AK68 operation are discussed.

## INTRODUCTION

While teaching a sophomore level introductory course on microprocessors to the Penn State Altoona electrical engineering technology students, the authors of this paper observed that a significant amount of students’ laboratory time was spent on an aspect of the exercises considered to be extraneous to the fulfillment of the laboratory objectives. The laboratory experience centered on elemental computer programming with a hardware emphasis. To accomplish this, exercises were devised involving a Motorola M68000 based small board computer and a corresponding commercially available cross-assembler.

Although a major aspect of the laboratory experience involved CPU programming, the actual experience was observed to be highly skewed toward another activity. A significant amount of time was spent learning the environment in which the programming was performed. Although learning the programming environment, as

opposed to learning programming, is a necessary activity, the authors deemed that it should be a minor activity and quickly mastered.

The development of the Motorola M68000 assembler-simulator software described in this paper was an attempt to reduce the time spent learning a particular programming environment, thereby increasing the time spent learning the 68000 programming language.

The excessive time required to master the assembler environment was tied to four main factors: (1) The lack of an integrated text editor resulting in a large number of separate program executions in order to obtain one error-free assembly. (2) The inability of the assembler to recognize a label unless it followed an unnecessarily strict set of identification rules. (3) The lack of a default data size parameter. (4) A limited set of cryptic error messages which failed to adequately inform the user if errors were generated as the result of 68000 language syntax requirements or arbitrary assembler rules of operation.

Additionally, it seemed desirable that the laboratory software should assist in a deductive resolution of correctly assembled but faultily executing code. This desire propelled the integration of a 68000 assembler, text editor, and simulator as described below.

## THE AK68 ASSEMBLER/SIMULATOR: THEORY OF DESIGN

The AK68 assembler/simulator was designed to assemble and then simulate error-free code until that point at which an error occurs. There is an important distinction to be made between this mode of operation and the more exotic design mode which is intended to continue to execute beyond executable code.

The difference between the two design philosophies is that AK68 informs the operator that the program has crashed; the exotic design informs the operator after some time, given the random bit patterns that RAM has been reduced to, exactly how the CPU continues to 'trash' the remaining memory. Although the latter is a real consideration in the design of 68000 based computers, it is not necessary for the more modest requirements needed to learn how to write a 68000 assembly program. In other words, AK68 is designed for writing 68000 programs, it is not designed to identify hardware-software failures.

In particular, the more modest AK68 design executes multiple EXCEPTIONS in a non-68000 manner. This difference is not visible when running normal 68000 assembly language programs, but becomes apparent if an attempt, by design or otherwise, is made to execute exceptions inside exceptions. At some point a non-68000 simulator generated "DOUBLE EXCEPTION" error will occur and the program counter (PC) will be set to the value contained at Memory address $0004 (the Motorola "RESET INITIAL PC" value).

This probably would not be the way the same code would act when running on a 68000 based board even with the BIOS EPROMS disabled. The result is that simple EXCEPTION processing routines can be written and executed within the simulator, but in-depth testing of these routines cannot be performed.

The EXCEPTION processing is not different from standard 68000 operation in the manner in which memory is manipulated. The vector table is emulated; the status register is stacked at the SSP location in accordance with 68000 rules of operation. The non-68000 nature of the simulator's operation appears within the exact time at which the exception is processed. AK68 does not emulate an internal multi-phase CPU clock: Multiple simultaneous exceptions (more than one exception occurring during the execution of a single instruction), particularly in the presence of TRACE exception generation and privileged instructions which change the SR register, result in exception processing which may or may not be equivalent to the order of execution which would occur in a real 68000 hardware environment.

The lack of the ability to prioritize single instruction multiple exceptions and the resultant inability to track the convolutions of multiple errors is a reasonable trade off for the benefits derived: A small and reasonably fast program simulator that faithfully runs all programs as long as they run. When they stop running, the simulator indicates a failure and resets the program counter (PC). In some environs this may be a limiting factor, but for learning how to write 68000 programs, this may in fact be an advantage.

## SOFTWARE DESIGN

AK68 consists of six major programs and a pop-up hexadecimal/binary/decimal calculator. Each major program operates from within one of three 'windows'. Each window is identified by a bracketed number found at the top right corner of the screen as seen in figures 1 through 5. The program is written in Borland Turbo Pascal [TM] version 6.0 and is based on Borland's Turbo Vision object oriented library which has been placed in the public domain by Borland.

Window [1]: This window contains the following two programs:
(1) The text editor: The text editor is a standard ASCII text editor which recognizes pointer operations, such as a mouse, and all alpha-numeric keys, the insert, delete, home, end, page-up, and page-down keys. It also recognizes the control sequence '<Ctrl>Y' which deletes the entire line at the cursor position. The maximum file size is 64000 characters including white space which translates to approximately 800 lines of assembly code. The text editor is the default condition while in Window [1], that is, when not performing any menu selected operation, then the text editor is active. Figure 1 is a DOS "Print Screen" output of a typical 68000 code fragment. Note that the original is in DOS Mode Color 80 allowing the top and bottom menu lines, and their various menu items, to be discriminated from the text editor by color.

(2) The 68000 assembler: The assembler operates from Window [1] and is activated by mouse-clicking on the highlighted word 'Assemble', or alternately by typing <Alt>A. (See Figure 1, top line.) The word "Assembler" blinks red during the two-pass assembly operation. An error-free compilation is then indicated by the bracketed block at the top left corner of the screen turning into the ubiquitous smiley-face symbol.

An incorrect assembly is halted at the first occurrence of an error. This was found superior, given the design objectives, to generating a (typical) plethora of errors and overwhelming the student programmer. When the first error is encountered, the program scrolls the screen to the line which contains the error, and then, either highlights the entire line, or when possible, highlights the specific element (label, operator, operand, or sized operation) which generated the error. An error message, including corrective action, when such action can be anticipated, is displayed on the bottom line of the screen.

The Help program which is larger than the 68000 assembler, is included in the assembler. Its objective is to find and highlight a given error and then display a useful error message. The Help program also includes a menu driven reference file which lists all 68000 commands (indexed by Math, Logic, Shifts, Bit Diddles, Big Diddles, Compares, Jumps, Branch_On, Addr/Stack, Moves, Flags, BCD, and Other Stuff), sized operations (divided into Inherently Sized, Multiply Sized, Index Sized, Illegal Size, and Sized Assembler Commands), effective addressing modes, registers, number formats, assembler commands, flags, down loads, labels, the simulator, and a generic Help Me file.

To obtain an error-free assembly, the only assembler command required to be in every text listing is the 'END' command. All commands after the END statement are ignored. Labels are identified by the assembler by being the first word of a line and not being a legal operator. No special symbol or column placement is required. In addition to 68000 commands, some labels are made illegal simply to avoid confusion when such labels are subsequently used as operand constants or offsets, e.g. D8.

The assembler routes its output to Window [2] following a successful assembly. The word 'Assembler' stops blinking, the smiley-face symbol appears at the top right corner of the screen, and the hexadecimal code listing is found in Window [2]. Figure 3 is an example of a typical hex listing.

Window [2]: Window [2] contains the following three programs:
(1) In addition to being a simple text viewer, Window [2] contains the AK68 Stepper-Tracer. The Stepper operates when a successful assembly has been performed in Window [1] resulting in a hex listing in Window [2]. When the above conditions have been met, then the Stepper is the default program in Window [2].

A ‘HiLite’ bar will appear across the current program listing line. Hitting any key which does not designate a menu item causes the program to execute one instruction of the 68000 listing. If a 68000 Exception occurs or if the program attempts to execute code outside of the assembled program listing, then the HiLite bar will disappear and a half second beep will be produced.

The Stepper/Tracer is useful for tracking expected program flow, expected loop iterations, and in determining the instruction which generates an unexpected Exception Error. The Stepper can be operated concurrently with the Simulator described below. That is, a program can be stepped to a desired position and then Window [3] (described below) can be accessed to continue program flow or simply determine CPU register and memory values at that time. Note that although the simulator described below can execute any random memory contents (limited by the discussion above, see THE AK68 ASSEMBLER/SIMULATOR THEORY OF DESIGN), the Stepper can only operate upon a successfully assembled hex listing.

The Stepper which is an alternate view of the simulator described below, is an independent program from the assembler. Some simulators, for ease of construction, simulate code by operating upon the mnemonics and plane text contained in the original assembled listing. This mode of operation never allows a self-check of the assembler/simulator integration. Once an error enters the assembly stream (as a part of its hex output file), it is perpetuated by the simulator, and until compared to an actual hardware run, may never be noticed by the operator. Separately constructed and separately operating programs will immediately indicate to the operator that the assembler/simulator program itself is the cause of the error by the observation that a particular assembler command is at variance with the corresponding mnemonic or symbol contained in the hex output file.

Rather than operating on the mnemonics and plane text output of the assembler, the simulator converts the assembled hex output into a binary stream and then simulates the central processor unit by performing Boolean mathematical operations on that binary data in a manner similar to the action of a digital circuit.

(2) Window [2] also contains an RS232 serial communication program. The communication program is not a standardized function of AK68. It is a custom include file added to AK68 which enables communication between AK68 and a specific small board computer. When the communications module is present then hex listings from Window [2] may be directly down loaded to the intended computer. Other functions such as reading the small board computer’s memory contents are subject to the availability of command/response procedures present in the small board computer’s basic input-output routines.

(3) S-record writer. Selecting ‘Write HEX’ from the File Menu in Window [2] generates a standard S-record formatted file in the current directory. Most small board computers are supplied with a download program which can communicate an S-record file to the small board computer. A successful ‘Assembly’ must have been accomplished before the S-record writer can operate.

Window [3]: Window [3] contains the following major program:
Window [3] is the 68000 Small Board Simulator. (see figure 4). This simulator visually represents all of the 68000 internal registers, stack pointers, and the status register. Each register is modifiable during program execution. Data register display and modification may be selected to be either in decimal, hexadecimal, standard (and IBM extended code) ASCII, Binary Coded Decimal or 32-bit position binary. The program counter and address registers are always displayed and modified in hexadecimal and the status register is always displayed and modified in binary. In addition, a pop-up screen is available which translates the current status register’s contents in terms of 68000 branch instructions in both signed and unsigned formats (see figure 5, line 2).

Any part of the 64K memory bank can be selected for viewing or modification in hex. The memory display sub-window contains the 128 byte block most recently read or written by the program being executed. A different block may be recalled by mouse-clicking on any of the block addresses and entering a new address in hex.

From the header menu a number of stepping operations may be chosen. Selecting ‘Run to Count’ will set the number of instructions to execute before the next break. Setting the count number to 0 will turn the function off, that is, the simulator will run continuously. Selecting ‘Run to Address’ will set an address break. Selecting a negative

number, or hitting <Esc>, will turn the 'Run to Address' function off. The 'Run to Count' and 'Run to Address' functions may be set to operate concurrently. Setting 'Run to Address' off (the default condition) and setting 'Run to Count' to 1 (also the default) offers simple single stepping operation.

Since the simulator is effectually running the 68000 in a pseudo-trace mode, turning the simulator's status register's Trace Mode bit 'On' causes the simulator to 'trace' through the 68000's trace mode stacking and vector table addresses, that is, it 'traces' the trace mode.

To track current operation, the last memory location and the last central processor unit's register acted upon is always highlighted. The last instruction performed is displayed on the next to the bottom line on the screen (see figure 4, "Last>MOVE.B DO,D2"). The next instruction to be executed is displayed on the bottom line of the screen (see figure 4, "Next> AND.L #$0000007,D2").

## FUTURE ENHANCEMENTS

A 6820 PIA software emulator acting through the personal computer parallel printer port will be added. The I/O emulation will add a limited hardware ability to the simulator so that lab exercises can be operated with external inputs and outputs up to the parallel port's size limit.

## REFERENCES


1. M68000 Family Programmer's Reference Manual, M68000PM/Ad, Motorola, Inc., 1989.
2. M68000 8-/16-/32-Bit Microprocessors User's Manual, Eighth Edition, Prentice Hall, Inc., Englewood Cliffs, NJ 1990.
3. The 68000: Principles and Programming, Blacksburg Continuing Education Series, by Leo J. Scalon, Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1981.
4. Microprocessor and Microcomputer Data Digest, by Walter H. Buchsbaum and Gina Weissenberg, Reston Publishing company, Inc., Reston, Virginia, 1983.
5. Turbo Pascal Version 6.0, Scotts Valley, Calf., Borland International.


## BIOGRAPHY

SOHAIL ANWAR - Sohail Anwar obtained a Ph.D. in Industrial and Vocational Education from The Pennsylvania State University in December 1995 and an M.S. degree in Electrical Engineering from the University of Texas at Arlington in May 1982. He completed additional graduate coursework in control theory and applied mathematical sciences at the University of Texas at Arlington during 1982-1984. Since August 1992, Sohail has been working as an assistant professor of engineering technology at Penn State Altoona.

DUREL HAUSER - Durel Hauser obtained Associate of Applied Science degrees in electrical engineering technology and in computer engineering technology from The Pennsylvania State University in 1995. Durel is currently working for the US Postal Service in Dubois. Previous experience includes service in the Navy as a Fire Control Technician aboard the Nuclear submarine U.S.S. Dallas where he operated and maintained a complicated mainframe computer system.

ROY SHEEHAN- Roy Sheehan is a graduate of the US Army Electronics School, Ft. Gordon, GA. with a specialty in cryptographic machines. He has additional training in pencil beam radar ranging equipment with the Air National Guard where he was the chief enlisted maintenance sergeant for the mobile unit. In his civilian life, for the past twenty-two years he has been involved in the design, installation, and maintenance of electronic systems for local industry and in his owner-operated business, Sheehan Electronics, Ashville, PA. Roy has a number of computer programs and micro-processor systems in current use in industry and university settings.