Motorola™ 68000 Family Simulators in Education

Susan A. Mengel and James M. Conrad

INTRODUCTION

Implementations of microprocessors are becoming more and more complex and difficult to understand for the student beginning to learn how to design computer hardware and to program in assembly language. Certainly, simple microprocessors exist and may be used to ease student learning toward more complex microprocessor designs. Still, microprocessors are tiny black boxes that cannot be fully explored without the aid of a computer that could magnify its inner workings.

In like manner, the design of computer hardware is becoming more and more complex. For example, digital circuit design involves a high level of detail and work to make sure all of the pins on a chip are connected correctly and the connections do not interfere with each other (crosstalk).

To help the student understand microprocessors, the student can write and run assembly language programs, and view the microprocessor's registers along with external memory through the use of a debugger. The student can get a clear idea of how the microprocessor goes about executing a program, but cannot view the inner detail of executing each instruction.

To help the student understand digital circuit design, computer-aided design packages have been developed at varying levels of sophistication. The student, however, must deal with a large amount of detail before he/she may have the cognitive grasp of how to do the digital circuit design.

The assembler's debugger cannot help the student with digital circuit design and the circuit board design package cannot help the student learn assembly. What is needed is a package to allow the student to design a digital circuit with less detail, such as allowing the student to connect chips on a board without worrying about pin outs and chip location problems. To enable the student to understand the chips or microprocessors he/she has connected, the package can show the student how program instructions are executed and what happens on the circuit upon execution. The simulator, further, can show levels of detail of the microprocessor's inner workings to provide the student with a higher or lower level view of how the microprocessor works.

In order to facilitate the student's understanding about microprocessors and digital circuit design, a graphical user interface should be employed to help the student visualize how microprocessors work and how they may be connected together to form a simple or even a sophisticated computer. A visual interface can lend concreteness to lessons on microprocessors and digital circuit design.

Currently, the Computer Systems Engineering Department offers a course where a digital circuit is designed with the Motorola 68000 microprocessor (CSEG/ELEG 4983 Computer Hardware Design). In order to enhance the laboratory exercises for the students and to allow them to experiment with more than one microprocessor, a simulator is under development which will allow the student to choose among the microprocessors in the 68000 family (giving the student the opportunity to increase the level of complexity in the microprocessor) and to place them into digital circuit designs. The student can get an idea of what he/she wants before using the computer aided design package to complete the design of the circuit.

DEBUGGERS

As mentioned before, debuggers will not allow a student to see how a microprocessor interacts with other chips on a digital circuit when a program is being executed. The features a debugger[2, 9], however, may have for assembly language debugging include: setting a process' starting point; setting breakpoints; examining variables; starting, restarting, and stopping a process; stepping through code; printing memory and registers; and displaying the source code. The debugger may even have a graphical user interface where the source code that is being executed is displayed, the registers are displayed, and the stack and data portions of memory are shown. On the other hand, a debugger, in general, will not allow the source code to be changed nor all portions of memory to be modified even if modification of the CPU registers might be allowed.

DIGITAL CIRCUIT DESIGN PACKAGES

Several digital circuit design packages are available for Computer Aided Design/Computer Aided Engineering. These packages provide the capability to specify, design, and simulate a digital design, even down to the electrical characteristics of signal wires placed near each other on a printed circuit board. Very often these packages provide considerably more function and require more detail than is desired in an undergraduate course. For example, a course in computer organization and design may be concerned with the design and simulation of a circuit, but not concerned about placement and routing of components on a printed circuit board. In fact, often the instructor is more concerned with the "logical" connections of the design than the physical point-to-point connections.

As an example, suppose one wants to design a computer architecture with a 68000 CPU, 32 Kbytes

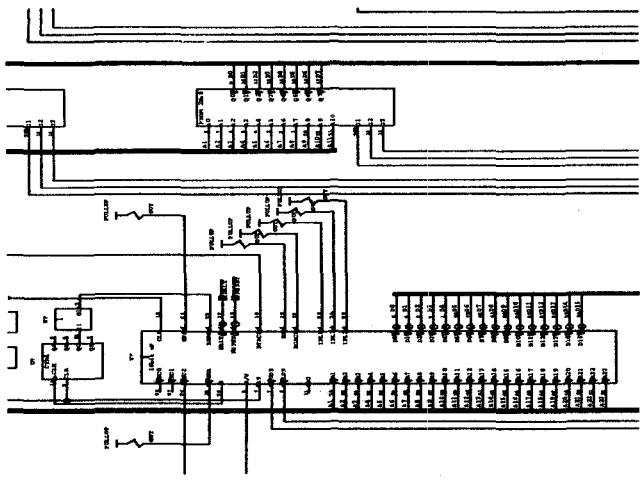

Fig. 1. Detail of a Motorola 68000 and ROM.

of memory, and a single serial communication chip (like the ACIA 6850 chip). Using a package like Powerview^{TM}[7] from ViewLogic^{TM}, one would need to design memory and I/O device decode logic, then connect the correct control, address, and data pins to each other as shown in Figure 1. Often all that is desired is to identify the memory map of the devices and to see how they interact with each other based on the device characteristics.

MOTOROLA SIMULATOR DESIGN

A search of the Internet was made in order to find a suitable Motorola simulator. A couple of bug-ridden 68000 simulators were found which were not useful for expansion to the more complex 68000 family chips. Fortunately, due to one of the author's former affiliation with North Carolina State University, a 68000 simulator, named 68KSIM[6], with an X Windows user interface[5] was obtained. The simulator ran on a DEC^{TM} workstation, but was easily modified to run on a SUN workstation. It also was used by students in a large introductory course so it had been tested well.

Upon examination of 68KSIM, it became apparent that errors existed in the code and that the user interface could be improved. The code, however, once corrected could be used as a basis for expansion.

Expansion of the simulation consisted of several steps: adding addressing modes and registers, implementing simulation memory and breakpoints, adding additional instructions and their routing, adding exception processing, and performing testing and verification of code. A more complete description of how the 68040 portion of the simulator has been implemented may be found in [3].

Interface Design

The user interface for the 68040 is designed to allow the student control over the execution of an assembly language program. The buttons on the interface are as follows:

- File: The user may load an assembly language program in S-record format, save the current state of the simulation, or exit the simulator.
- Run: The user may run the assembly language program.
- Stop: The user may stop the simulator to view the contents of memory or the registers.
- Step: The user may step through the assembly language program one instruction at a time and examine the contents of the registers or memory.
- Exception: The user may set exception flags or schedule interrupts. To schedule interrupts, the user fills in a pop-up window of 16 interrupts with priority levels and program address locations. For example, when an interrupt with a level above zero is specified, that interrupt will be scheduled to occur when the program counter reaches the associated address.
- Memory: The user may specify the size of memory in Kbytes or fill a block of memory with a specified value.
- Breakpoint: The user may set up to 16 breakpoints through a pop-up window.
- Clear: The user may clear the contents of all registers.

The main window of the interface has five control areas:

1. The registers of the Motorola 68040 microprocessor: the A0 through A7 address, the D0 through D7 data, the User Stack Pointer (USP), and Status (SR) registers.
2. Sixteen registers that are used in supervisor mode.
3. The Program Counter and Cycle Counter registers.
4. Memory - A 22x16 display of bytes with 22 address fields and the following buttons: page up or down one memory page (352 bytes), move up or down one memory line (16 bytes), immediately display the first memory window (starting at address zero), immediately display the last memory window, and immediately display a user specified starting address.
5. The instruction cache lines (currently under development).

The main window, popup windows, buttons, menus, and text fields in the interface were designed using OpenWindows^{TM} Developer's Guide[8] and implemented with XView^{TM}[4].

```
1000            MOVE.W #$AA,$1120   ;VARIOUS MOVE OPERATIONS
1006            MOVE.W #$05,D0
100A            MOVE.W #$04,D1
100E            MOVE.L #$1034,D2
1014    LOOP    MOVE.W D0,$1100     ;LOOPING EXAMPLE
1018            SUBI #1,D0
101A            BNE LOOP
101C            BKPT #0             ;BREAKPOINT WILL STOP SIMULATOR
101E            MOVE16 (A0),(10C4).L ;68040 INSTRUCTION
1024            LSR                 ;ILLEGAL ADDRESSING MODE USED HERE
1026            MOVEC D2,VBR        ;68040 SUPERVISOR INSTRUCTION
102A            ILLEGAL             ;EXCEPTION HANDLING EXAMPLE
102C            BKPT #0
102E            MOVE #$FFFF,D0      ;SAMPLE EXCEPTION HANDLER
1032            RTE
1034                                ;EXCEPTION VECTOR TABLE BEGINS
.
.
.
1044            0000102E            ;VECTOR 4: ILLEGAL INSTRUCTION
```

Fig. 2. Example Assembly Code Fragment.

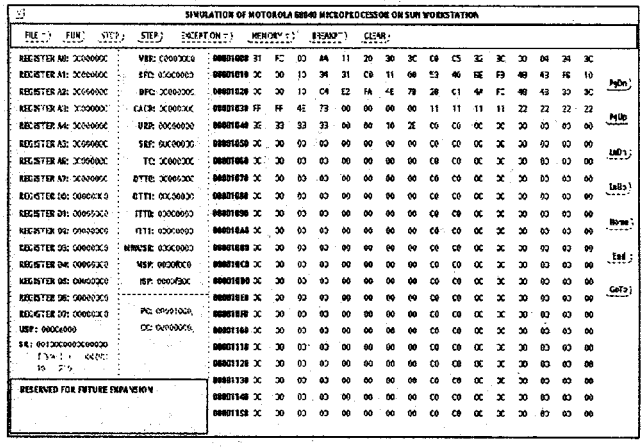

Fig. 3. Assembly Code Fragment Loaded.

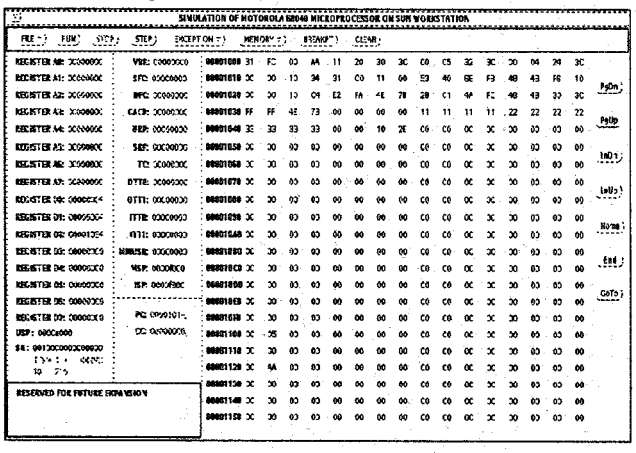

Fig. 4. Simulator After One Iteration of the Loop.

Example Session

The example session is designed to show the reader of this paper how the user would interact with the 68040 interface. The program in Figure 2 which is an assembly language code fragment will be loaded into the simulator and executed. This program's only use is to show some of the features of the simulator.

The assembly code fragment is loaded into memory starting at address 1000h as shown in Figure 3. The program counter has been changed manually to point to address 1000h. Now the user has the option of stepping through the program one instruction at a time (STEP button) or simply running the program (RUN button).

In Figure 4, the user has used the STEP button to step through the program down to one iteration of the loop located from address 1014h to 101Ah. The last instruction executed is the BNE LOOP so the program counter now points to 1014h which is at the beginning of the LOOP. At this point, the user selects the RUN button to allow the simulator to execute the program until the simulation halts.

The simulation halts at the BKPT instruction. A notification box pops up and the user has the option of

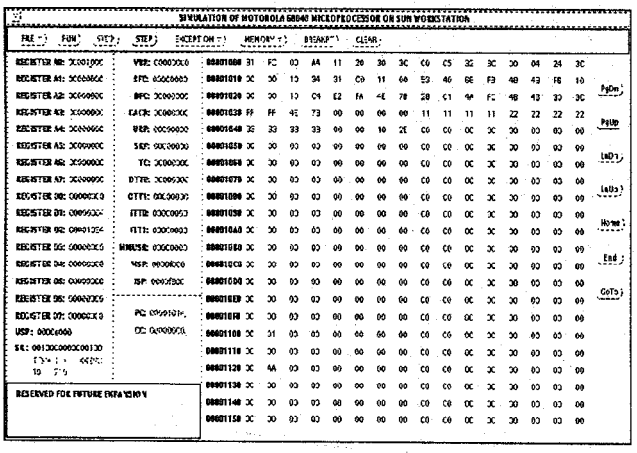

Fig. 5. BKPT Encountered.

continuing or halting the program execution. At this point, halt is chosen and register A0 is modified manually to 1000h so that the MOVE16 instruction does not cause an error since it uses register A0. Figure 5 shows the resulting simulation screen. The user may now use the STEP or RUN button to execute the program.

The MOVE16 instruction executes next where A0 has

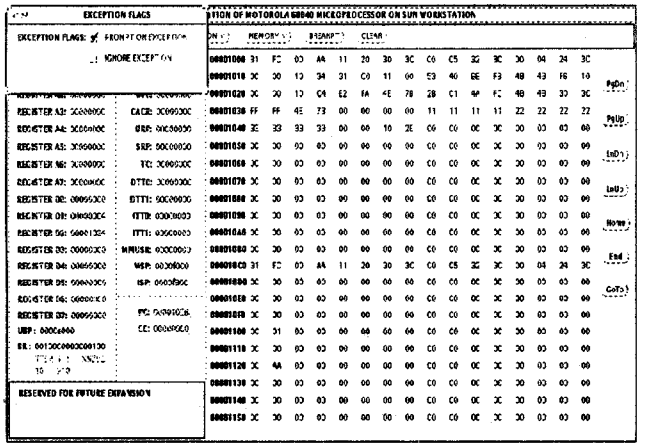

Fig. 6. Turning On Exception Handling.

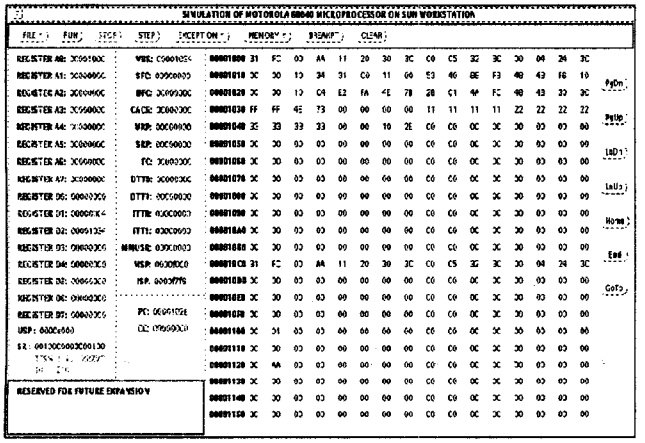

Fig. 7. ILLEGAL Instruction Exception Handler.

a value which causes the first 16 bytes of the program to be copied to address 10C4h. Then the LSR instruction is encountered causing an exception. Even though a notice is popped up telling the user of the exception, the exception is ignored since exception handling is not enabled through the EXCEPTION menu at this time. The user can enable exception handling as shown in Figure 6 by turning off Ignore Exceptions.

After the MOVEC instruction which initializes the exception vector base register, the ILLEGAL instruction is encountered forcing an illegal instruction exception to occur. The user is given a notification of the exception and the option of continuing or halting. If the program continues to execute, it will look at the address associated with vector 4 in the exception vector table. The simulator will jump to the address in that table entry and execute the exception handler. The address located in vector 4 is 102Eh where a small exception handler is located. Figure 7 shows the program counter pointing to address 102Eh. Figure 8 shows the four word stack frame placed upon the interrupt stack located at address F7F8h at the top of the screen.

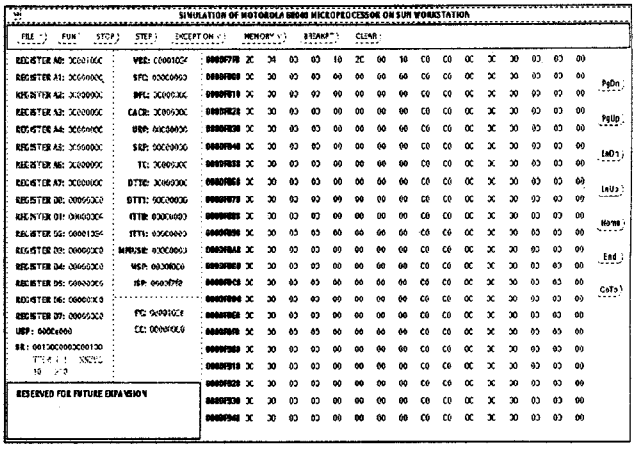

Fig. 8. Interrupt Stack Frame.

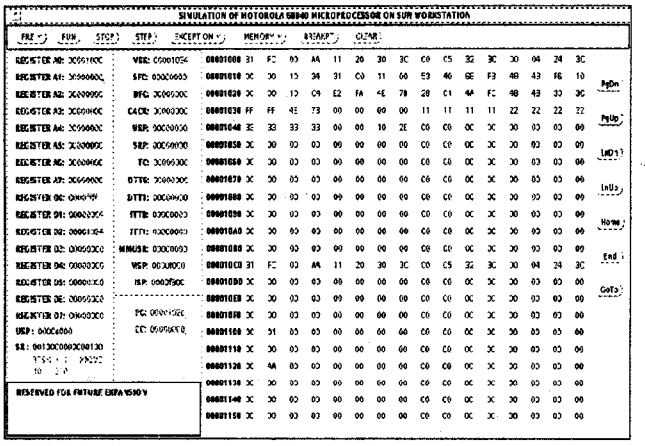

Fig. 9. Final Screen.

Finally, the RTE from the exception routine is executed. Then, the final instruction executes which is another BKPT. Now, the user halts the program. The results are shown in Figure 9.

COMPUTER HARDWARE SIMULATOR

The computer hardware simulator is under design and will incorporate a bus architecture where the student will be able to pick and choose which components to connect to the bus. A small representation of the entire computer designed by the student may be given on the interface where different components are highlighted that are communicating during the execution of an assembly language program. The student will be able to enlarge a component if desired to see what it is doing.

From the above description, the computer hardware simulator will require:

- Additional chip simulators for interrupt controllers, serial ports, parallel ports, and disk controllers

- A design interface to allow the student to construct a digital circuit including address decode logic (based on boolean equations), memory mapping of I/O, and timing specifications
- An extension to the Motorola simulator to allow an interface to the computer hardware simulator

FUTURE WORK

The Motorola 68000 family of microprocessors was chosen in support of course material and research being conducted at the University of Arkansas in multichip modules[1]. Even though it would be good to include other microprocessors, the simulator may become larger than desired for efficient operation. Instead it might be better to incorporate a "generic" microprocessor where the student can place his/her own desired functionality into the microprocessor to enable the student to learn how to design microprocessors. The components of the generic microprocessor might represent the best and the worst of current microprocessors. A generic microprocessor simulator might be easier to maintain than trying to keep up with a family of microprocessors.

Along with the generic microprocessor, the student might be given the capability to design an assembly language specifically for the microprocessor. The student could choose from a menu of general assembly language instructions or design his/her own instructions.

ACKNOWLEDGEMENTS

The following individuals helped in the implementation of the 68040 simulator: Trey Grubbs, Bill Herring, Richard Tan, David Andrews, Carl Bowling, and Ronald Goforth.

Motorola is a registered trademark of Motorola, Inc. X Window System is a product of the Massachusetts Institute of Technology. Sun Workstation is a registered trademark of Sun Microsystems, Inc. XView and OpenWindows are trademarks of Sun Microsystems, Inc. DEC is a trademark of Digital Equipment Corporation. Powerview and Viewlogic are registered trademarks of Viewlogic Systems, Inc.

NOTE: A copy of this paper with larger figures may be obtained via anonymous ftp from: `engr.engr.uark.edu`, `/user/sam/fie94-msim.ps.Z`.

REFERENCES

[1] David L. Andrews, James M. Conrad, Leonard Schaper, Susan Mengel, and Daniel J. Berleant. "Design of a High Speed MIMD Distributed Processor Node Using MCM Technology." *Proceedings of the 1993 International Electronics Packaging Conference*, International Electronics Packaging Society, Wheaton, IL, pp. 132-139.

[2] *Debugging Tools.* Mountain View, CA: Sun Microsystems, 1988.

[3] Trey Grubbs, Bill Herring, Richard Tan, and Susan Mengel. "Motorola 68040 Microprocessor Simulation for the SUN Workstation^{TM}." *Applied Computing 1994, Proceedings of the 1994 Symposium on Applied Computing*, ACM Press, New York, NY, 1994, pp. 25-30.

[4] Dan Heller. *XView Programming Manual Volume Seven.* Sebastopol, CA: O'Reilly and Associates, Inc., 1990.

[5] Jay Lloyd. *68KSIM Simulation Interface.* North Carolina State University, 1990 (for more information, contact Dr. James Conrad at jmc3@engr.uark.edu).

[6] Tan Phan. *68KSIM.* North Carolina State University, 1989 (for more information, contact Dr. James Conrad at jmc3@engr.uark.edu).

[7] *Powerview.* Viewlogic Systems, Inc., 293 Boston Post Road West, Marlboro, MA 01752-4615.

[8] *Openwindows Developer's Guide 3.0 User's Guide.* Mountain View, CA: Sun MicroSystems, 1991.

[9] *Turbo Debugger 3.0 for Windows.* Scotts Valley, CA: Borland International, 1991.

*****

Susan A. Mengel

Susan A. Mengel received her Ph.D. in Computer Science from Texas A&M University in 1990. She joined the Computer Systems Engineering Department at the University of Arkansas in Fayetteville in 1991 as an Assistant Professor. She became an Adjunct Assistant Professor in the Electrical Engineering Department at UAF in 1993. She has worked in the areas of student modeling with intelligent tutoring systems and neural networks. Other research interests include user modeling and computer-assisted instruction. She has consulted in the areas of neural networks and expert systems. She has served on the Steering Committee of the Artificial Intelligence in Education Society. She may be reached via the internet at sam@engr.uark.edu.

James M. Conrad

James M. Conrad received his B.S. Degree in Computer Science from the University of Illinois-Urbana/Champaign in 1984. He received his M.S. and Ph.D. in Computer Engineering from North Carolina State University in 1987 and 1992, respectively. He worked for IBM from 1984 to 1990, and is currently an Assistant Professor in the Computer Systems Engineering Department at the University of Arkansas (email: jmc3@engr.uark.edu). His research interests include computer architecture, parallel programming, and engineering education. He is a member of the ACM, Eta Kappa Nu, IEEE, IEEE Computing Society, and AAAI.